CNN Compression with Tucker Decomposition

Eric Ho
University of California - San Diego
erho@ucsd.edu

Abstract

As deep learning continues to improve, more CNNs are being deployed on mobile devices. As a result, there is a constraint due to limited computational power and memory capacity due to the millions of parameters in CNNs. This paper addresses this problem and proposes a solution by compressing deep networks using Tucker Decomposition. By factorizing the weights of the convolutional layers, the model becomes more lightweight and faster in terms of inference. There is a 65% reduction in computations from the model used in experimentation.

1 Introduction

It is well known that deep neural networks have an absurd amount of parameters which help the model learn the internal representations of the dataset at hand. These parameters allow the model to converge to a local minimum of the loss function. Although this is not an issue for high end computers that have GPUs and countless amount of storage capacity, portable and mobile devices do not have such luxury. Many techniques have tried to solve the problem of making the model more lightweight and reducing computational complexity while maintaining or even improving its accuracy. Pruning and dropout are effective methods of doing so. Another alternative is proposed in this paper which is to use Tucker Decomposition to decompose the pre-trained weight matrix into smaller matrices. As a result, computational complexity of the model is reduced, making the model more lightweight[3]. Although this technique can be applied to any layer, we will only be decomposing convolutional layers.

2 Methods

2.1 Tucker Decomposition

The Tucker decomposition decomposes a tensor into a smaller core tensor and a set of matrices [1]. The tensor can be of any size. To better visualize the Tucker decomposition, we will first consider the Tucker decomposition in 3 modes or dimensions as shown in figure 1.

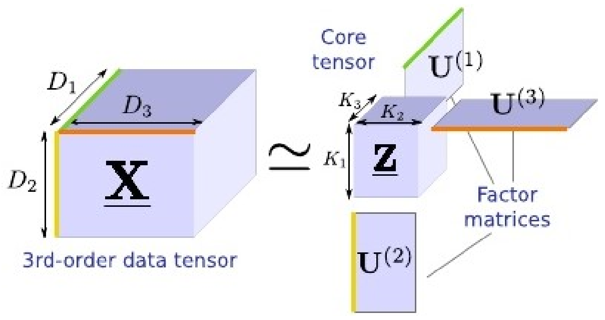

Figure 1: Three mode Tucker decomposition.

The Tucker decomposition in figure 1 can be formulated as

$$X = Z \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \tag{1}$$

where $\times_n$ is a mode-n product. The original tensor $X \in R^{D_1 \times D_2 \times D_3}$ is decomposed to the core tensor $Z \in R^{K_1 \times K_2 \times K_3}$ and matrices $U^{(1)} \in R^{D_1 \times K_1}$, $U^{(2)} \in R^{D_2 \times K_2}$, and $U^{(3)} \in R^{D_3 \times K_3}$.

Convolution layer weights are essentially a 4-way tensor $W^{k \times k \times c \times f}$ where k is the filter size, c is the number of input channels, and f is the number of output channels. By performing Tucker decomposition on W, there will subsequently be one core matrix and 4 matrices, effectively converting a convolution into four smaller convolutions.

In the experiments, we will only decompose W in the c and f directions which becomes into three convolution layers. The formula now becomes

$$W = core \times_3 I \times_4 O \tag{2}$$

where $core \in R^{k \times k \times R1 \times R2}$, $I \in R^{c \times R1}$ and $O \in R^{R2 \times f}$. The ranks $R1$ and $R2$ can be modified depending on the desired compression of the weights. In order to correctly perform the three convolutions the the padding of the convolutional layers must be the same as the inputs. The first convolution layer will have a stride and kernel size of 1. As for the weights, it will use I for the kernel and have no bias. The second convolution layer will use the $core$ tensor as the kernel weights and zero bias. It will have the same stride, kernel size, and activation layer as the original convolution layer. Lastly, the third convolution layer will use the O tensor as the weights and the bias from the original convolution layer. It also has a stride and kernel size of 1.

2.2 Computational Complexity

In general, the computational complexity of a convolutional layer is $k \times k \times c \times f \times m \times n$ where m and n is the size of the input, assuming a stride of 1 and padding such that the output has the same size as the input. By performing Tucker decomposition on the weights, there will be more convolutional layers but a decrease in the number of computations. Specifically, it takes $m \times n \times c \times R1$ for the first convolution layer, $m \times n \times k \times k \times R1 \times R2$ for the second layer, and $m \times n \times R2 \times f$ for the last layer which comes out to be a total of $(m \times n)(c \times R1 + k \times k \times R1 \times R2 + R2 \times f)$.

2.3 Variational Bayesian Matrix Factorization

As mentioned above, ranks $R1$ and $R2$ can be chosen arbitrarily, however, it can also be found deterministically. Variational Bayesian Matrix Factorization (VBMF) is such way to find $R1$ and $R2$ to be used in the Tucker decomposition. Essentially variational Bayesian (VB) approximation is used when the calculation of the Bayes posterior is difficult [2]. Using VB for matrix factorization enables automatic dimensionality selection that is useful for applications such as compression.

3 Experiments

In the following section, we will explore the usefulness of Tucker decomposition on convolutional layers. The dataset we used was the CIFAR-10 dataset. We used Adam as the optimizer and the sparse categorical cross entropy loss function from Keras.

3.1 CNN Model

The model we used for retraining and evaluating is the following:

Layer type	Parameters
Conv2D	1792
ReLu Activation	0
BatchNormalization	265
Conv2D	36928
ReLu Activation	0
BatchNormalization	265
Max Pooling	0
Conv2D	73856
ReLu Activation	0
BatchNormalization	512
Conv2D	147584
ReLu Activation	0
BatchNormalization	512
Max Pooling	0
Conv2D	295168
ReLu Activation	0
BatchNormalization	1024
Conv2D	590080
ReLu Activation	0
BatchNormalization	1024
Flatten	0
Dense	131584
ReLu Activation	0
BatchNormalization	2048
Dense	262656
ReLu Activation	0
BatchNormalization	2048
Dense	5310
Softmax	0

It has a total of 1,552,458 parameters and has a test accuracy of 90%.

3.2 Varying *R1* and *R2*

The first experiment deals with the information lost when performing Tucker decomposition. Focusing on only on decomposing the second convolution layer, we first kept $R2 = 64$ and varied $R1 \in [1, 64]$. We then reconstruct the weights using 2 and calculate the error $||W - \bar{W}||^2$, where $\bar{W}$ is the reconstructed weight. We do this process again but keeping $R1 = 64$ and varied $R2 \in [1, 64]$. The results are shown in figure 2

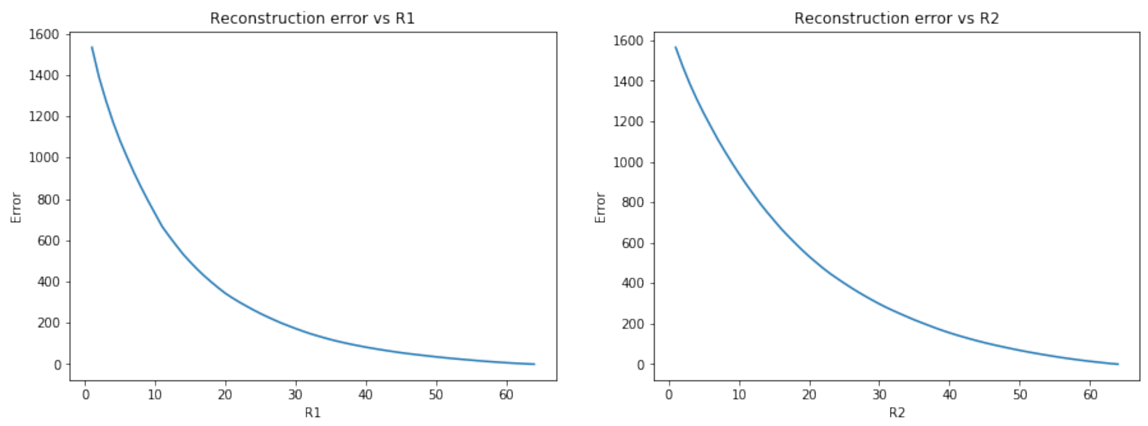

Figure 2: (Left) The reconstruction error keeping R2=64. (Right) The reconstruction error with R1=64.

For both cases, we can see that as the rank increases to 64, the original rank, the reconstruction error decreases to nearly zero. This makes sense since there is no information lost in the Tucker decomposition. On the other hand, if the rank is small, the approximation becomes more and more inaccurate.

3.3 Applying Tucker Decomposition

Next we will apply Tucker decomposition on all convolutional layers of our model except the first convolution layer and measure the test accuracy before and after training for 10 epochs. As for the ranks, we set $R1 = c \cdot x$ and $R2 = f \cdot x$ where $x \in [\frac{1}{8}, \frac{1}{2} ... \frac{7}{8}]$.

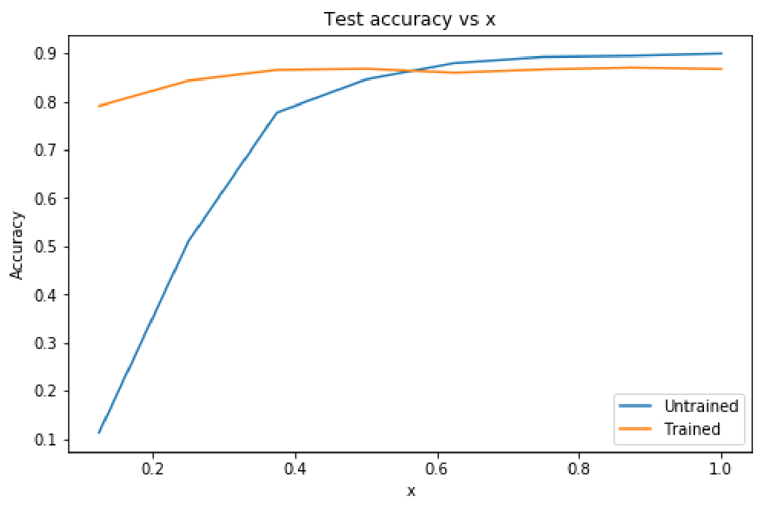

Figure 3: Plot of test accuracy before and after training the decomposed model.

From figure 3, we see that before training, the test accuracy deteriorates for small values of x. However, after training the model, the accuracy is above 80% regardless of x. In general, not a significant amount of test accuracy is sacrificed after decomposing the model after training. Moreover, the point where training actually performs worse than without training is around $x = 0.58$. At that point, the computation complexity is 900,000 which reduces the complexity by nearly 42%! Figure 4 shows the complexity of the decomposed model.

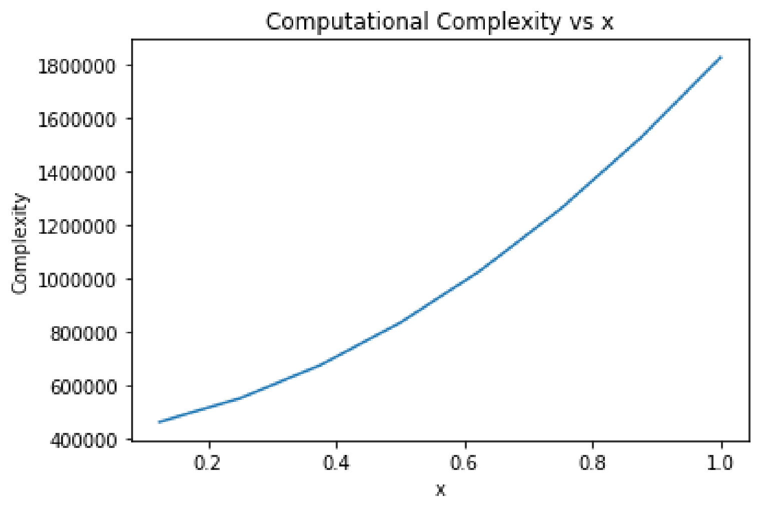

Figure 4: Plot of computational complexity of the decomposed model as various x values.

One interesting observation is when $x = 1$ which results in the original ranks of the convolutional layers. At that point, the computational complexity is 18,000,000 which is 3 million more than the original model. Although the test accuracy does not decrease nor improve, we have unnecessarily more parameters. This means that using Tucker decomposition can actually make the model *worse* depending on the value of the ranks.

3.4 Tucker Decomposition with VBMF

Lastly, we analytically compute the ranks using VBMF for each convolutional layer and then decompose the model. The test accuracy is shown in figure 5. In the end, the model obtained 87.5% accuracy which is 3% lower than the original test accuracy. However, it achieved a computational complexity of 530,291 which is 65% less than the original!

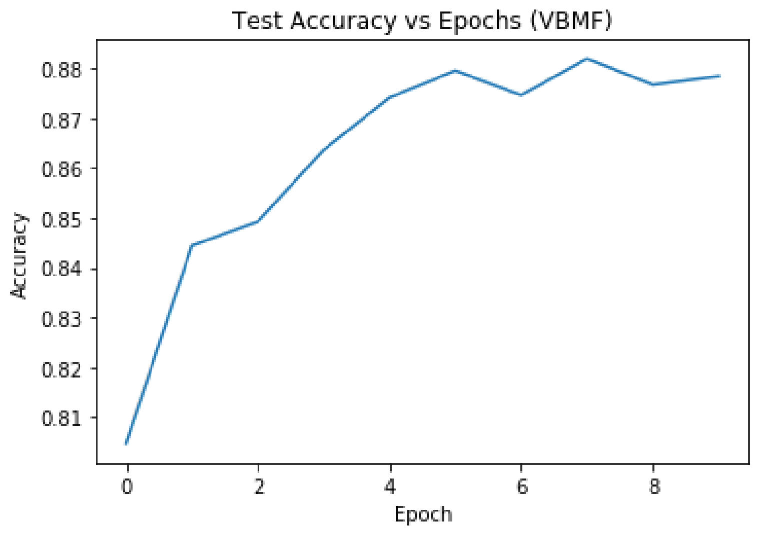

Figure 5: Test accuracy of the model with ranks obtained using VBMF.

4 Conclusion

Tucker decomposition can significantly compress a neural network. In our experiments, we have only decomposed the convolutional layers, however, this technique can be used on any layer with weights. In the first experiment, we looked at the reconstruction error with various ranks. We found that the error increases exponentially as rank is decreased. In the second experiment, we saw the effect of retraining the decomposed model and was able to reduce the computational complexity by more than 50% while maintaining decent test accuracy. In the last experiment, we calculated the ranks for Tucker decomposition using variational Bayes matrix factorization and saw a 65% reduction in computational complexity while sacrificing only 3% test accuracy.

References

[1] Tamara G. Kolda and Brett W. Bader. *Tensor Decompositions and Applications*. http://www.ca.sandia.gov/~tgkolda/pubs/bibtgkfiles/SAND2007-6702.pdf, 2007.

[2] S. Babacan R. Tomioka S. Nakajima, M. Sugiyama. *Global Analytic Solution of Fully-observed Variational Bayesian Matrix Factorization*. http://www.jmlr.org/papers/volume14/nakajima13a/nakajima13a.pdf, 2013. Accessed: 2018-06-08.

[3] Sungjoo Yoo Taelim Choi Lu Yang Yong-Deok Kim, Eunhyeok Park and Dongjun Shin. *Compression of deep convolutional neural networks for fast and low power mobile applications.* https://arxiv.org/abs/1511.06530.