Neural Network Compression using Transform Coding and Clustering

Anonymous Author(s)

With the deployment of neural networks on mobile devices and the necessity of transmitting neural networks over limited or expensive channels, the file size of trained model was identified as bottleneck. We propose a codec for the compression of neural networks which is based on transform coding for convolutional and dense layers and on clustering for biases and normalizations. With this codec, we achieve average compression factors between 7.9–9.3 while the accuracy of the compressed networks for image classification decreases only by 1%–2%, respectively.

1 Introduction

Deep neural networks spread to many scientific and industrial applications. Often, the necessity of large amounts of training data, long training duration and the computational complexity of the inference operation are noted as bottlenecks in deep learning pipelines. More recently, the memory footprint of saved neural networks was recognized as challenge for implementations in which neural networks are not executed on servers or in the cloud but on mobile devices or on embedded devices. In these use cases, the storage capacities are limited and/or the neural networks need to be transmitted to the devices over limited transmission channels (e.g. app updates). Therefore, an efficient compression of neural networks is desirable. General purpose compressors like Deflate (combination of Lempel-Ziv-Storer-Szymanski with Huffman coding) perform only poorly on neural networks as the networks consist of many slightly different floating-point weights.

In this paper, we propose a complete codec pipeline for the compression of neural networks which relies on a transform coding method for the weights of convolutional and dense layers and a clustering-based compression method for biases and normalizations. Our codec provides high coding efficiency, negligible impact on the desired output of the neural network (e.g. accuracy), reasonable complexity and is applicable to existing neural network models, i.e. no (iterative) retraining is required.

Several related works were proposed in the literature. These works mainly rely on techniques like quantization and pruning. The tensorflow framework provides a quantization method to convert the trained floating-point weights to 8 bit fixed-point weights. We will demonstrate that considerable coding gains on top of those due to quantization can be achieved by our proposed methods. Han et al. proposed the Deep Compression framework for the efficient compression of neural networks. In addition to quantization, their method is based on an iterative pruning and retraining phase. In contrast to Deep Compression, we aim at transparent compression of existing network models without the necessity of retraining and without modifying the network architecture. It is known from other domains like video coding that transparent coding and coding modified content are different problems. Iandola et al. propose a novel network architecture called SqueezeNet which particularly aims at having as few weights in the network as possible. We will demonstrate that our method can still reduce the size of this already optimized SqueezeNet network by a factor of up to 7.4.
2 Compression Method

In this section, we describe our complete codec design with reference to the pipeline illustration in Fig. 1. The trained neural network model is the input of the codec. All layers of the network are processed individually. This simplifies partly retroactive updates of individual layers without transmitting the complete network again. Weights, biases and normalizations are pre-scaled to use the complete dynamic range of the number representation.

It is observable that the filters in neural networks contain structural information not completely different from blocks in natural pictures. Reasoned by this observation, the encoder base for convolutional filters consists of a two-dimensional discrete cosine transform (2D DCT) followed by a quantization step. This combination is often referred to as transform coding.

For the DCT, the transformation block size is set accordingly to the size of the filter (e.g. a $7 \times 7$ DCT for a $7 \times 7$ filter). Subsequent to the transformation, the coefficients are quantized. The bit depth of the quantizer can be tuned according to the needs of the specific application. Typical values are 5-6 bit/coefficient with only a small accuracy impact.

The weights of dense layers (also referred to as fully-connected layers) and of $1 \times 1$ convolutions (no spatial filtering but filtering over the depth of the previous layer, typically used in networks for depth reduction) are arranged block-wise prior to transform coding.

K-means clustering is used for the coding of the biases and normalizations. The number of clusters is set analogously to the quantizer bit depth according to the quality settings. Code books are generated for biases and normalizations. Thereby, the usage of the clustering algorithm is beneficial if less bits are needed for coding the quantizer indices and the code book itself than for coding the values directly. The clustering approach has the advantage that the distortion is smaller than for uniform quantization. In consequence, the accuracy of the network is measured to be higher for a given number of quantizer steps. However, the occurrence of code book indices is also more uniformly distributed. Due to the higher entropy of this distribution, the compression factor is considerably smaller (see Sec. 3). In particular the Burrow-Wheeler transform and the move-to-front transform which are both invoked for entropy coding are put at a disadvantage by the uniform distribution. We chose to use the same number of quantizer steps for all parameters. For this reason the clustering was chosen for those network parameters which are too sensible to the higher distortion caused by uniform quantization.

The processed data from the transform coding and from the clustering are entropy coded layer-wise using BZip2, serialized and written to the output file. In addition, meta data is stored. It includes the architecture of the layers in the network, shapes and dimensions of the filters, details on the block arrangements, scaling factors from the pre-scaling, scaling factors and offsets from the quantizer, and the code books for the clustering.

3 Evaluation

We evaluate our method using four image classification networks: ResNet50 (2), GoogLeNet (3), AlexNet (9) and SqueezeNet (8). Rate-distortion analysis is a typical procedure for the evaluation of compression algorithms (10). The performance of neural networks for image classifi-
We analyze the computational complexity of our algorithms by measuring the encoder and decoder run times using our unoptimized Python code. For the final codec (transform coding for convolutional and dense weights, clustering for biases and normalizations), we measure 29s for the encoder and 7.6s for the decoder.

4 Conclusion

In this paper, we proposed a codec for the compression of neural networks which is based on transform coding and clustering. The codec enables a low-complexity and high efficient transparent compression of neural networks. The impact on the neural network performance is negligible.
References


