**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2019/m51018**

**October 2019, Geneva, Switzerland**

|  |  |
| --- | --- |
| **Source:** | **InterDigital Communications, Inc.** |
| **Status:** | **Input contribution** |
| **Title:** | **[NNR] Conditional arithmetic coding for quantized parameters** |
| **Authors:** | **Shahab Hamidi-Rad, Fabien Racapé, Swayambhoo Jain,** |

# Introduction

This contribution presents an entropy coding method in which quantized tensor data are broken down to header and body. Adaptive arithmetic coding is then used for serialized header and conditional arithmetic coding is used for the indexes (body).

# Proposed method

We use conditional arithmetic coding for the body of the quantized information which according to Shannon's source coding theorem results in best compression ratio.

According to “Shannon's source coding theorem” [2], the average optimal code length for symbols produced by a probability model is given by the entropy. Assuming a code book with k entries with probability pi for the ith entry, we have:

Since arithmetic coding doesn’t compress one data symbol at a time, it can get very close the entropy (e.g. optimal code length). By contrast, Huffman coding cannot reach entropy unless the probabilities of all symbols are powers of 2.

Arithmetic coding works best when the probability distribution of the symbols is known. Fortunately, for the case of compressing deep neural networks, we have a quantization stage preceding the entropy coding stage. When using the K-means based quantization, the probability distribution is automatically available as a byproduct of quantization process. As we know the slow compression/decompression is one of disadvantages of Arithmetic coding. The availability of probability model can help increase the speed of compression/decompression.

Since compression/decompression of DNNs is usually an offline process, in most cases the slowness of compression/decompression is acceptable, and the gain of significant compression ratio improvement justifies the slower process.

## Adaptive Arithmetic Coding

Arithmetic Coding works with a probability model. In most real applications the probability model, (the probability of each symbol happening in the data), is not available and varies during the compression process. Adaptive Arithmetic Coding creates a probability model based on occurrences of symbols in the data stream and updates this model continuously during the process. This is especially useful when a probability model is not available for the compressed data.

## Conditional Arithmetic Coding

Conditional Arithmetic Coding is another method to provide the Arithmetic Coding with a real-time probability model. In the beginning of the process it is assumed that we have exact number of occurrences for each codebook symbol in the tensor data. The probability of occurrence at this stage for each symbol is proportional to its number of occurrences in the data. As the data is consumed, the counts change, and the probability of occurrences is not the same as the start of process. This can be explained as conditional probabilities. In other words, at each stage we use the original probability conditioned by the counts of each symbol already consumed.

Conditional Arithmetic Coding is expected to provide the best compression ratio because it uses the most accurate probability model at any time during the compression/decompression process.

## Compressing Layers of Deep Neural Networks

Deep Neural Networks are made up of several layers. Each layer has a set of parameters that are obtained during the training process. These parameters are stored as tensors (multi-dimensional matrixes). We compress the DNN parameters one tensor at a time.

For each tensor we get the following information as the output of quantization process:

1. Shape: The information about the dimensionality of the tensor. A 2-D, 64x32 weight matrix for example is represented by the list [64, 32]. As another example, a 4-D weight tensor for a convolution layer with kernel size 3x3 and input and output channels 16, 32, can be represented by [3, 3, 16, 32]
2. Codebook: This is a set of floating-point values. (Centers of clusters calculated by the K-Means Quantization algorithm) The data symbols for entropy coding are the indexes to the codebook values.
3. Symbol Counts: This is a list of integer values with same length as codebook. The ith item in this list gives the number of times the ith codebook symbol occurs in the tensor.
4. The indexes: This is a list with a length equal to the size of original tensor. (i.e. the length of the flattened version of original tensor) Each item in this list is an index to the item in the codebook which is closest to the corresponding value in the original tensor.

We first serialize the shape, codebook, and symbol counts information into an array of bytes. This is the “header” part of the information being compressed. The header is compressed using adaptive arithmetic coding.

The indexes are usually the largest part of information and can contain millions of integer values. We call this the “body” of the information being compressed. The body is compressed using conditional arithmetic coding using the Symbol Counts as initial probability model and updating it as symbols are encoded.

The decoder receives and decodes the header first. It then can use the initial probability model to decode the body. The indexes (the decoded body) are reshaped to the original shape using the shape information and passed back to the dequantization stage together with the codebook.

## Serializing and encoding Header Data

The shape is stored as an array of integer values. The floating-point values in the codebook are assumed to be 32-bits in IEEE-754 format. Each floating-point value is serialized to a list of 4 bytes.

Before the compression, we first make sure that the entries in the codebook are sorted based on the frequency of occurrences of each symbol. This means the first item in the codebook corresponds to the most probable symbol. This also means that the Symbol Counts list is a list of integers in descending order.

The symbol count values can be very large numbers. Instead of storing these large numbers in the header, we store the difference of each item with its previous one. In most cases this results in smaller numbers in the list which reduces the size of array.

After the serialization, we have all header information as a list of bytes. This is given to adaptive arithmetic coding algorithm. As explained before, adaptive arithmetic coding does not require a probability model as it automatically create an initial model (with equal probability for symbols) and updates it as each symbol is seen in the data.

# Results

In the following experiments we first reduced the number of parameters by applying Low Rank Decomposition on different layers of the network. Then we applied K-Means quantization with different codebook sizes. The results of quantization for each tensor of each layer were passed to the Entropy coding method explained above to create the compressed bitstream.

The network was then reconstructed from the bitstream and the performance was tested on the test data set and the results are reported as the Reconstructed Accuracy.

## Image classification

## VGG16

|  |  |  |
| --- | --- | --- |
| Original Network | # Params | 138,357,544 |
| O\_Size(4x # Params) | 553,430,176 |
| Top-1 | 0.7062 |
| Top-5 | 0.89736 |
| Reduced # ParamsApplied LR Decomposition (CE1) | 34633512 |
| Codebook Size | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| Quantized Size (CE2) | 12,988,647 | 17,318,884 | 21,649,973 | 25,982,702 | 30,318,139 | 34,658,444 | 39,009,488 |
| Encoded Size (Cs\_size) | 11,089,989 | 14,505,214 | 17,953,904 | 21,930,559 | 26,181,814 | 30,394,897 | 34,538,334 |
| ReconstructedAccuracy | Top-1 | 0.41446 | 0.67456 | 0.7025 | 0.70282 | 0.70642 | 0.7079 | 0.70788 |
| Top-5 | 0.65938 | 0.87614 | 0.88924 | 0.8876 | 0.88876 | 0.88938 | 0.88924 |
| Compression Ratio (Cs\_size/O\_size) | 0.0200 | 0.0262 | 0.0324 | 0.0396 | 0.0473 | 0.0473 | 0.06240 |

## End-to-end image compression

|  |  |  |
| --- | --- | --- |
| Original Network | # Params | 76,179 |
| O\_Size(4x # Params) | 304,716 |
| PSNR | 30.137648 |
| Reduced # ParamsApplied LR Decomposition (CE1) | 53,523 |
| Codebook Size | 128 | 256 | 512 | 1,024 | 2,048 |
| Quantized Size (CE2) | 63,612 | 77,644 | 96,392 | 168,152 | 194,232 |
| Encoded Size (Cs\_size) | 54,541 | 67,676 | 85,131 | 155,902 | 180,633 |
| Reconstructed PSNR | 28.282715 | 29.921356 | 30.588062 | 30.788263 | 30.788378 |
| Compression Ratio (Cs\_size/O\_size) | 0.1789 | 0.2220 | 0.2793 | 0.5116 | 0.5927 |

## Audio Classification

|  |  |  |
| --- | --- | --- |
| Original Network | # Params | 116,815 |
| O\_Size(4x # Params) | 467,260 |
| Accuracy | 0.582716 |
| Reduced # ParamsApplied LR Decomposition (CE1) | 47,607 |
| Codebook Size | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| Quantized Size (CE2) | 24,868 | 31,773 | 39,151 | 46,444 | 54,924 | 66,708 | 83,072 |
| Encoded Size (Cs\_size) | 23,925 | 31,243 | 38,318 | 45,432 | 53,521 | 64,674 | 79,598 |
| Reconstructed Accuracy | 0.577778 | 0.590123 | 0.596296 | 0.602469 | 0.604938 | 0.604938 | 0.604938 |
| Compression Ratio (Cs\_size/O\_size) | 0.0512 | 0.0668 | 0.0820 | 0.0972 | 0.1145 | 0.1384 | 0.1703 |

## Comparing with DeepCABAC

The following table compares the results of compression for 4 different tensors from VGG16 and LeNet-5 networks. These tensors were first quantized using different quantization codebook sizes and then the results were given to two different entropy coding algorithms (DeepCABAC [2] and conditional arithmetic coding).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tensors to compress** | **Layer Type** | **Tensor Shape** | **No. Indexes (Body)** | **Codebook Length** | **Compressed No. of bytes** |
| **DeepCABAC** | **Conditional Arithmetic Coding** |
| 3rd Layer of LeNet-5 (L3\_FC) | Fully Connected | 400x120 | 48,000 | 32 | 30,701 | 29,356 |
| 4th Layer of VGG16 (S2\_L2\_CONV) | Convolution | 3x3x128x128 | 147,456 | 226 | 134,838 | 131,457 |
| 13th Layer of VGG16 (S5\_L3\_CONV) | Convolution | 3x3x512x512 | 2,359,296 | 309 | 2,340,349 | 2,230,261 |
| 15th Layer of VGG16 (S6\_L2\_FC) | Fully Connected | 4096x4096 | 16,777,216 | 63 | 12,327,325 | 11,701,723 |

Here are the network structures of LeNet-5 and VGG16:

**LeNet-5**

Scope InShape Comments OutShape Activ. Post Act. # of Params

------------- ------------ ----------------------- ------------ ------- --------------- -----------

L1\_CONV 28 28 1 KSP: 5 1 s 14 14 6 Tanh MP(KSP):2 2 s 156

L2\_CONV 14 14 6 KSP: 5 1 v 5 5 16 Tanh MP(KSP):2 2 v 2,416

L3\_FC 5 5 16 120 Tanh 48,120

L4\_FC 120 84 Tanh 10,164

L5\_FC 84 10 Softmax 850

---------------------------------------------------------------------------------------------------------

 Total Number of parameters: 61,706

**VGG16:**

Scope InShape Comments OutShape Activ. Post Act. # of Params

------------- ------------ ----------------------- ------------ ------- --------------- -----------

S1\_L1\_CONV 224 224 3 KSP: 3 1 s 224 224 64 ReLU 1,792

S1\_L2\_CONV 224 224 64 KSP: 3 1 s 112 112 64 ReLU MP(KSP):2 2 s 36,928

S2\_L1\_CONV 112 112 64 KSP: 3 1 s 112 112 128 ReLU 73,856

S2\_L2\_CONV 112 112 128 KSP: 3 1 s 56 56 128 ReLU MP(KSP):2 2 s 147,584

S3\_L1\_CONV 56 56 128 KSP: 3 1 s 56 56 256 ReLU 295,168

S3\_L2\_CONV 56 56 256 KSP: 3 1 s 56 56 256 ReLU 590,080

S3\_L3\_CONV 56 56 256 KSP: 3 1 s 28 28 256 ReLU MP(KSP):2 2 s 590,080

S4\_L1\_CONV 28 28 256 KSP: 3 1 s 28 28 512 ReLU 1,180,160

S4\_L2\_CONV 28 28 512 KSP: 3 1 s 28 28 512 ReLU 2,359,808

S4\_L3\_CONV 28 28 512 KSP: 3 1 s 14 14 512 ReLU MP(KSP):2 2 s 2,359,808

S5\_L1\_CONV 14 14 512 KSP: 3 1 s 14 14 512 ReLU 2,359,808

S5\_L2\_CONV 14 14 512 KSP: 3 1 s 14 14 512 ReLU 2,359,808

S5\_L3\_CONV 14 14 512 KSP: 3 1 s 7 7 512 ReLU MP(KSP):2 2 s 2,359,808

S6\_L1\_FC 7 7 512 4096 ReLU 102,764,544

S6\_L2\_FC 4096 4096 ReLU 16,781,312

S6\_L3\_FC 4096 1000 Softmax 4,097,000

---------------------------------------------------------------------------------------------------------

 Total Number of parameters: 138,357,544

# Test frameworks and platform

The presented results were obtained under the following test conditions:

* **Operating System:** Ubuntu 18.04 LTS
* **Programming Language:** Python 2.7, Tensorflow 1.11.0
* **Additional Libraries:** Cuda v9.0.176 and CUDNN 7.2
* **The hardware configuration:**
	+ Intel i9-7920x CPU@2.9 Ghz
	+ NVIDIA TITAN Xp
	+ 128 GB DDR4 RAM

# Patent rights declaration(s)

**InterDigital Communications, Inc. may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# References

1. “Description of Core Experiments on compression of neural networks for multimedia content description and analysis”, ISO/IEC JTC1/SC29/WG11 N18574, July 2019.
2. Simon Wiedemann et. Al. “Response to the Call for Proposals on Neural Network Compression: End-to-end processing pipeline for highly compressible neural networks” ”, ISO/IEC JTC1/SC29/WG11 m47698, March 2019.