**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2020/m54469**

**June 2020, OnLine**

|  |  |
| --- | --- |
| **Source:** | **InterDigital Communications, Inc.** |
| **Status:** | **Input contribution** |
| **Title:** | **[NNR] CE4: Combined Quantization and Arithmetic Coding** |
| **Authors:** | **Shahab Hamidi-Rad, Swayambhoo Jain, Fabien Racapé** |

# Introduction

This contribution reports InterDigital’s results to the Core Experiment 4 [1], which is a combination of codebook quantization as explained in [2], [3] and conditional/adaptive arithmetic coding as explained in [4]. In addition to combining the two methods, a few modifications have been made to the methods which are explained in sections 2 and 3.

Note that the following results reflect the performance of the quantization without requiring access to annotated data, i.e. without the gradient-based clustering described in [2]. Sections 2 and 3 describe the proposed modifications to the original method related to the bounded PDF, section 4 includes the suggested changes to the high-level syntax for the new codebook information, and section 5 details the test results.

# Modifications to Quantization:

This section explains the modifications made to the quantization method explained in [2] and [3]. Please note that all the modifications are only minor encoding-side changes and do not change the main behavior of the codebook quantization.

## Minimizing Entropy using Bounded-PDF initialization

The parameter “pdfFactor”, as explained in [3], has a value between 0.0 and 1.0 and defines how much the encoder relies on the PDF-based feature. If it is zero, PDF based initialization is not used at all. In this case the K-Means algorithm is initialized uniformly. A value of 1.0 means the encoder uses PDF feature with its maximum effect.

Since the quantized model goes through further compression at entropy coding stage, the encoder aims to minimize not only the error between the quantized and original tensor, but also the entropy of the quantized model. When using Arithmetic Coding, the entropy of a quantized tensor is a good estimation of its size after entropy coding.

The entropy of each quantized tensor can be calculated using the probability model (counts of symbols) which is available as an output of codebook quantization. Probability of a symbol “s” is given by:

P(s) = count(s)/totalCount

Where count(s) is the number of times the symbol “s” appears in the indexes of a quantized tensor and totalCount corresponds to the sum of count(s) values for all symbols. The entropy of the tensor after quantization is then:

In the new proposed implementation, instead of receiving pdfFactor as a parameter for the quantization, the PDF factor that minimizes the entropy of the quantized tensor is selected.

In our experiments this not only reduces the size of the final bitstream, but in most cases it also results in better accuracy and consistently improved the rate/distortion quality of the compressed model.

## Binary search for best symbol count

In the original implementation detailed in [2] and [3], we used to receive as input parameters the minimum and maximum “qBit” values. This restricted the number of symbols (codebook size) to be a power of 2. We then used to do a linear search to find the best “qBits” given a specific MSE.

In the new implementation, min and max qBit values are not required anymore. The new method runs a binary search to find the best symbol count (codebook size) for a given MSE value.

In current implementation, the binary search is done over the range of 4 to 4096 (equivalent of qBits 2 and 12) for the codebook size.

## Uniform Codebook quantization

Codebook quantization works best with larger tensors. For smaller tensors, the overhead of codebook size makes the compression inefficient. To solve this problem, a “Uniform Codebook” is proposed. The Uniform Codebook only includes a few integer values that can be used to reconstruct the codebook at the decoder. In this case the codebook entries are spaced evenly, and the results of quantization is very similar to a uniform quantization. Similar to the actual codebook quantization, we do a binary search to find the best symbol count (which is equivalent of step size in uniform quantization) for a given MSE value.

## Uniform quantization of codebook entries

Instead of including the floating-point values in the codebook, we now quantize them uniformly. With this change we don’t need to use any floating-point value in the quantized tensor data.

## Unified Codebook Information

We defined a unified Codebook format that covers both regular and uniform codebook types in addition to a couple of special cases. This is an array of integers that replaces the codebook defined in [2] and [3].

The first integer in the array specifies the type of codebook. The next entries in the codebook info depend on the codebook type. Please note that for the specific codebook types 1 and 2 below, we don’t need to include the actual quantized tensor in the bitstream. The codebook information in these cases contain all the information needed to reconstruct the tensor.

**Codebook Type 1:** All tensor entries have the same integer value.

This is a rare case that can happen for some tensors (Particularly in ResNet50)

Codebook Info: [1, intVal]

Where intVal is the integer value for all entries of the tensor.

**Codebook Type 2:** All tensor entries are *almost* the same floating-point value. This means if we did a uniform quantization of the tensor, all resulting integer values would be the same.

This is a rare case that can happen for some tensors (Particularly in ResNet50 and MobileNetV2 networks)

Codebook Info: [2, rangeInt, symCount, offset]

On the decoder side, the floating-point value used for all entries of the tensor can be calculated as follows:

step = float(rangeInt)/(symCount-1)

floatVal = offset \* step

**Codebook Type 3:** Uniform codebook as explained in 2.3 above.

Codebook Info: [3, rangeInt, symCount, offset]

On the decoder side, for each entry “q” in the quantized tensor (Integer), the corresponding entry “r” in the reconstructed tensor can be calculated as follows:

step = float(rangeInt)/(symCount-1)

r = (q + offset)\*step

For codebook types 2 and 3, on the encoding side, “symCount” denotes the symbol count for the conditional arithmetic coder, which is obtained by the quantization algorithm (Binary search based on the maxMSE). The values for “rangeInt” and “offset” are calculated using the original tensor as follows:

rangeInt = ceil(max(tensor)) – floor(min(tensor))

step = float(rangeInt)/(symCount-1)

offset = round( min(tensor)/step

**Codebook Type 0:** Regular codebook quantization.

Codebook Info: [0, cbRangeInt, cbSymCount, cbOffset, cbInt0, cbInt1, …, cbIntN-1 ]

In this case the cbInt0 to cbIntN-1 are the N entries of codebook quantized to integer values. On the decoder side, we first reconstruct the floating-point codebook and then use it to reconstruct (i.e. lookup) the tensor entries. Assuming rcbFloati is the floating-point reconstructed codebook entry corresponding to the integer cbInti:

cbStep = float(cbRangeInt)/( cbSymCount -1)

rcbFloati = (cbInti + cbOffset)\*cbStep

recCodebook = [rcbFloat0, rcbFloat1, …, rcbFloatN-1]

For each integer entry “q” in the quantized tensor, the corresponding floating-point entry “r” in the reconstructed tensor is:

r = recCodebook[ q ]

On the encoding side, the original codebook after K-mean quantization is:

floatCodebook = [cbFloat0, cbFloat1, …, cbFloatN-1]

The values for “cbRangeInt”, “cbSymCount”, and “cbOffset” can be set after codebook quantization of the tensor as follows:

cbRangeInt = ceil(max(floatCodebook)) – floor(min(floatCodebook))

cbSymCount = max(minCbSymCount, symCount2)

cbStep = float(cbRangeInt)/( cbSymCount -1)

cbOffset = round( min(floatCodebook)/step )

cbInti = round(cbFloati/cbStep) – cbOffset

where cbInti is the quantized integer value corresponding to the ith entry in the original floating-point codebook cbFloati. Current implementation uses minCbSymCount = 220.

# Modifications to Arithmetic Coding:

This section explains the modifications made to the conditional arithmetic coding as detailed in [4].

## Using adaptive arithmetic coding with large codebooks

For large codebooks, the overhead of probability model (the symbol counts array) can be significant. So, if the ratio between the size of tensor to the size of codebook is less than a threshold value, we use adaptive arithmetic coding (which does not need a probability model). In current implementation this threshold is set to 50.

## Using adaptive arithmetic coding for uniform codebook quantization

For uniform codebook quantization (Codebook Type 3), we always use adaptive arithmetic coding.

# Suggested changes to the bitstream

The following changes are suggested to include the new codebook information in the “nnr\_compressed\_data\_header” syntax element. The new parts are highlighted.

|  |  |
| --- | --- |
| nnr\_compressed\_data\_unit\_header( ) { | **Descriptor** |
| **nnr\_compressed\_data\_unit\_payload\_type** | u(5) |
| **nnr\_multiple\_topology\_elements\_present\_flag** | u(1) |
| **nnr\_decompressed\_data\_format\_present\_flag** | u(1) |
| **input\_parameters\_present\_flag** | u(1) |
| if (nnr\_multiple\_topology\_elements\_present\_flag == 1) |  |
| topology\_elements\_ids\_list() |  |
| else |  |
| **ref\_id** | st(v) |
| if (nnr\_compressed\_data\_unit\_payload\_type == NNR\_PT\_CB) |  |
| codebook\_info() |  |
| if (nnr\_decompressed\_data\_format\_present\_flag == 1) |  |
| **nnr\_decompressed\_data\_format** | u(7) |
| if (input\_parameters\_present\_flag == 1) { |  |
| **tensor\_dimensions\_flag** | u(1) |
| **cabac\_unary\_length\_flag** | u(1) |
| if (tensor\_dimensions\_flag == 1) |  |
| tensor\_dimensions() |  |
| If (cabac\_unary\_length\_flag == 1) |  |
| **cabac\_unary\_length** | u(8) |
| } |  |
| byte\_alignment() |  |
| } |  |

|  |  |
| --- | --- |
| codebook\_info() { |  |
| **codebook\_type** | u(v) |
| if ( codebook\_type == 0) { |  |
| **cbRangeInt** | u(v) |
| **cbSymCount** | u(v) |
| **cbOffset** | i(v) |
| **codebook\_size** | u(v) |
| for ( j = 0 ; j < codebook\_size; j++ ) { |  |
| **codebook\_int[j]** | u(v) |
| } |  |
| } |  |
| else if ( codebook\_type == 1) { |  |
| **tensor\_int\_value** | u(v) |
| } |  |
| else if ( codebook\_type == 2 || codebook\_type == 3) { |  |
| **rangeInt** | u(v) |
| **symCount** | u(v) |
| **offset** | i(v) |
| } |  |
| } |  |

The new codebook information contains only integer values that can be represented with any variable length method in the bitstream.

# Results

The following results are based on the following pipelines:

* Input Network à Codebook Quantization followed by Adaptive/Conditional Arithmetic Coding

**Input networks:**

* Original Networks
* CE1 outputs based on baseline Low Rank method [3] for working points WP1 and WP4 as detailed in [1]
* CE1 outputs based on baseline Sparsification method provided by Nokia (nokia2Plus results). Only results for MobileNetV2 and ResNet50 are available.

**Use cases:**

* VGG-16
* ResNet-50
* MobileNetV2
* DCase (Audio Classification)
* UC12B (Image Compression)

We modified NCTM [5] with the changes explained in sections 2 and 3. Figure 1 below shows the top-5 error as a function of the compressed model size for the proposed method “Interdigital (CE4)”, vs our previous method “Interdigital Codebook+Arithmetic” vs NNR baseline which corresponds to “HHI CE2+3 / 14” that was adopted at the last meeting. For complete results, please refer to the attached spreadsheets

A close up of a map

Description automatically generated

Figure 1: Comparison of top-5 error rate on MobilNet

# Test frameworks and platform

The presented results were obtained under the following test conditions:

* **Operating System:** Ubuntu 18.04 LTS
* **Programming Language:** Python 3.6, Tensorflow 1.14.0
* **Additional Libraries:** Cuda V10.0.130 and CUDNN 7.6
* **The hardware configuration:**
  + Intel i9-7920x [CPU@2.9](mailto: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 N19227, April 2020.
2. Shahab Hamidi-Rad, Swayambhoo Jain, Fabien Racapé “CE2: Report on Codebook Quantization with bounded PDF initialization” ISO/IEC JTC1/SC29/WG11 m53765, April 2020
3. Shahab Hamidi-Rad, Swayambhoo Jain, Fabien Racapé “Gradient-Based K-Means for Quantization of Deep Neural Network parameters” ISO/IEC JTC1/SC29/WG11 m49260, July 2019.
4. Shahab Hamidi-Rad, Fabien Racapé, Swayambhoo Jain “Conditional arithmetic coding for quantized parameters” ISO/IEC JTC1/SC29/WG11 m51018, October 2019
5. “Test model 4 of Compression of neural networks for multimedia content description and analysis”, ISO/IEC JTC 1/SC 29/WG11 N19226