

KRZYSZTOF WRÓBEL 
MICHAŁ KARWATOWSKI 
MACIEJ WIELGOSZ 
MARCIN PIETROŃ 
KAZIMIERZ WIATR 

COMPRESSING SENTIMENT ANALYSIS CNN MODELS FOR EFFICIENT HARDWARE PROCESSING

Abstract

Convolutional neural networks (CNNs) were created for image classification tasks. Shortly after their creation, they were applied to other domains, including natural language processing (NLP). Nowadays, solutions based on artificial intelligence appear on mobile devices and embedded systems, which places constraints on memory and power consumption, among others. Due to CNN memory and computing requirements, it is necessary to compress them in order to be mapped to the hardware. This paper presents the results of the compression of efficient CNNs for sentiment analysis. The main steps involve pruning and quantization. The process of mapping the compressed network to an FPGA and the results of this implementation are described. The conducted simulations showed that the 5-bit width is enough to ensure no drop in accuracy when compared to the floating-point version of the network. Additionally, the memory footprint was significantly reduced (between 85 and 93% as compared to the original model).

Keywords

natural language processing, convolutional neural networks, FPGA, compression

Citation

Computer Science 21(1) 2020: 25–41

1. Introduction

Natural language processing (NLP) is considered to be one of three main application domains of deep learning (along with image and video processing). Nowadays, there is an increasing need to utilize it on mobile devices for applications such as translation, voice-typing, or image-to-text converters. As the computing power of mobile devices increases, this enables them to replace personal computers for many tasks; however, their resources are still severely limited. Consequently, the memory footprint and computational burden reductions can bring significant benefits. It is worth noting that, despite an abundance of research efforts in deep learning architecture compression for image processing [2, 6, 19, 20], there are just a few projects intended for NLP neural architecture compression [1, 5, 10, 15].

This paper is a case study of applying common compression techniques for architectures designed for NLP. After pruning and quantization were applied, the model was deployed to the FPGA platform. This has allowed for examining the feasibility and efficiency of using FPGAs in a domain of embedded neural computations. The experiments (conducted on standard datasets) revealed a strong relationship between the size and structure of the datasets and the performance of the quantization and pruning methods used. The quantization and pruning impact on the accuracy of the analyzed neural architecture [9] was examined separately for each layer.

The paper is structured as follows. Section 2 provides an overview of the used CNN architecture, quantization, and pruning processes. Section 3 briefly covers the datasets used for the experiments. Section 4 describes the FPGA implementation details. Finally, Section 5 contains the results of the experiments, and Section 7 summarizes the contributions of this work.

2. Compression of Neural Networks

2.1. CNN-non-static

Convolutional neural networks (CNNs) are composed of neurons that have learnable weights and biases. Each neuron receives multi-dimensional vectors (tensors) as its input. These then perform a convolution operation using multi-dimensional filters. This is usually followed by pooling or non-linearity functions. We employed one of the most commonly used NLP models; i.e., **CNN-non-static** from [9]. The model uses pre-trained 300-dimensional word GloVe embeddings [14], which are fine-tuned during training. Unknown words are randomly initialized. Two window sizes were used (2 and 3), resulting in 2 parallel CNN layers with 128 filters each. The CNN layers outputs were concatenated and then connected to a dense (fully-connected) layer with 128 outputs. CNNs and dense layers have rectified linear units as activation functions. The last layer uses a softmax activation function in the case of many outputs and a sigmoid activation function in the case of binary classification. After each layer, a dropout was applied with a probability of 50%. The training data was

divided into mini-batches of a size of 128. The training involves Nadam [4] as an optimizing algorithm. The training lasts 25 epochs, and the best model is saved.

The layers of a CNN have neurons that generate output feature maps $y_i, i = 1 \dots N$ from input feature maps $x_j, j = 1 \dots M$ as follows:

$$y_i = b_i + \sum_{j=1}^M F_{ij} * x_j, \quad (1)$$

where F_{ij} are two-dimensional (2D) convolutional kernels of dimensions $H \times W$, $*$ represents the convolution operation, and b_i are the bias terms.

The number of multiply-accumulate (MAC) operations and cycles spent on the execution in a practical implementation is often used as the metric for the complexity of a CNN. Assuming each output feature map y_i has P elements (where P is equal to the height multiplied by the width of a given feature map), the total number of MAC calculations for a convolutional filtering operation is $MACs = PHWMN$. The quantization role is intended to reduce the complexity of each of these operations, while the goal of the network compression through pruning is to reduce the total number of operations.

2.2. Pruning

After training a neural model, we acquire a set of weights for each trainable layer. These weights are not evenly distributed over the range of possible values for a selected data format. As presented in Fig. 1, most weights are concentrated around zero; therefore, their impact on the resulting activation value is not significant. The mechanism of pruning removes those weights whose values are below a certain threshold level. The authors of [7] examined how pruning affects convolutional neural networks for image classification, showing little accuracy drop and a high compression ratio.

In our work, pruning is applied to the 1D convolution layer according to Alg. 1. As the weights are symmetrically distributed around 0, the threshold was calculated for negative values based on the absolute value. Weight distribution may differ depending on the dataset, network architecture, and training algorithm. As a result, checking the distribution of the weights before applying the pruning is an important step. The example from Fig. 1 shows that a threshold level of 0.02 is sufficient to remove around 50% of the weights. Depending on the specific network implementation, storing the weights may require huge amounts of memory; removing weights through pruning have a direct impact on lowering the storage requirements.

The pruning mechanism is especially effective in FPGA implementations. In standard CPUs, processing throughput improvement is not important since the multiply and accumulate operation is replaced by a conditional; on most modern processors, both of these operations are supported by the hardware. More-significant improvements can be achieved when the pruning is extensive enough to enable algorithms for sparse operations; however, such extreme pruning will likely have a considerable impact on the accuracy.

Algorithm 1 One-D convolution layer with pruning.

```

1: for  $outS$  in  $output\_size$  do
2:   for  $outFM$  in  $output\_feature\_maps$  do
3:     for  $inFM$  in  $input\_feature\_maps$  do
4:       for  $kerS$  in  $kernel\_size$  do
5:         if  $abs(weight[inFM][outFM][kerS]) \geq pruning\_threshold$  then
6:            $output(outS, outFM) += input(outS + kerS, inFM) * weight[inFM][outFM][kerS]$ 
7:         end if
8:       end for
9:     end for
10:     $output(outS, outFM) += bias[outFM]$ 
11:  end for
12: end for

```

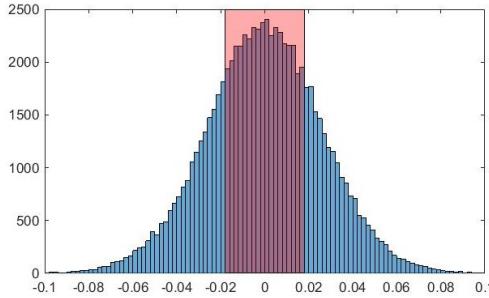


Figure 1. Histogram of the first convolutional layer of model weights – 50% of weights cut out by pruning operation (marked in red).

In an FPGA implementation of a neural network, however, it is possible to hard code weights into LUT or BRAM memories during synthesis; as a result, the total cost of resolving the condition whether the specific calculation should be performed is moved from inference to compilation. The weights that are removed by pruning will not be included in the synthesized design before the implementation step, saving both logic and routing resources. This technique, however, requires full kernel parallelization; for typical image processing architectures, it is not feasible to implement them in the current FPGAs. Network architectures for NLP have fewer dimensions and often use smaller kernels; therefore, kernel loops can be fully unrolled, even in mid-scale FPGA chips.

2.3. Quantization

Quantization is the procedure of constraining values from a continuous set or denser domain to a discrete set (e.g., integers) or sparser domain in which the quantized input

values will be represented. In our case, the domain is a floating-point representation. A floating-point number can be represented as follows:

$$\text{flp} = \mathbf{m} \cdot \mathbf{b}^{\mathbf{e}}, \quad (2)$$

where $\mathbf{m} \in \mathcal{Z}$ is the mantissa, $\mathbf{b} = 2$ is the base, and $\mathbf{e} \in \mathcal{Z}$ is the exponent. In the case of a single-precision floating-point format, the mantissa is assigned 23 bits, the exponent is assigned 8 bits, and 1 bit is assigned for a sign indicator (according to the IEEE-754 standard). Therefore, the set of values that can be defined by this format is described by the following:

$$\text{flp}_{\text{single}} : \{\pm 2^{-126}, \dots, \pm(2 - 2^{-23}) \times 2^{127}\}. \quad (3)$$

Similarly, the reduced precision IEEE-754 mini-float format assigns ten bits for the mantissa, five bits for the exponent, and one sign bit.

2.3.1. Dynamic fixed-point quantization

Currently, GPUs with parallel processing being applied to machine learning operate mainly using the single-precision format. In contrast, embedded DSPs and the latest GPUs operate using fixed-point processing, which restricts the numbers to a range:

$$\text{fpx} : 2^{-\text{frac_bits}} \cdot \{-2^{\text{total_bits}-1}, \dots, 2^{\text{total_bits}-1} - 1\}, \quad (4)$$

where $\text{total_bits} \in \{8, 16\}$ are conventional bit-widths, and frac_bits is a shift down (or up) that determines the fractional length (or the number of fractional bits) as well as integer length $\text{int_bits} = \text{total_bits} - \text{frac_bits} - 1$ for signed numerical representation. The format in which the fractional length varies over the individual coefficients or data samples is also referred to as *dynamic fixed-point*. The main difference between the traditional and dynamic formats is the fixed location of the integer and the fractional part separation in the former. This results in the utilization efficiency of the data format, which is higher in the dynamic fixed-point. However, dynamic fixed-point is more challenging in implementation; thus, traditional fixed-point prevails.

It is possible to define a general mapping from set of floating-point data $x \in \mathcal{S}$ to fixed-point q as follows (assuming signed representation):

$$q_{\text{fpx}} = Q(x_{\text{flp}}) = \mu + \sigma \cdot \text{round}(\sigma^{-1} \cdot (x - \mu)). \quad (5)$$

In our case, $\mu = 0$ and $\sigma = 2^{-\text{frac_bits}}$, where

$$\text{int_bits} = \text{ceil}(\log_2(\max_{x \in \mathcal{S}} |x|)) \quad (6)$$

and $\text{frac_bits} = \text{total_bits} - \text{int_bits} - 1$. Scaling factor σ is essentially just a shift up or down. One drawback is that a great deal of precision may be lost if a large mean skews the distribution of dataset \mathcal{S} .

A side effect of the adopted approach is representation saturation; this means that **int_bits** and **frac_bits** may not accommodate a full dynamic range of the original number representation. Some values denoted as outliers are saturated to the max or min of the quantization range. In order to determine how detrimental it is for the overall performance of the quantization module, a histogram analysis is conducted. This analysis determines the number of outliers that get truncated; these should amount to less the 0.1 % of the whole number of examined values in the examined dataset. In addition to this rule-of-thumb approach, some other methods were also examined; this quantization aspect is still open to further research.

2.3.2. Integer quantization

Another approach is quantization that maps floating-point values $x \in \mathcal{S}$ to integers:

$$q_{\text{int}} = \mathcal{Q}(x_{\text{flp}}) = \text{ceil} \left(\frac{(x - \mu) \cdot \sigma}{\max \mathcal{S} - \min \mathcal{S}} \right). \quad (7)$$

The μ parameter can be set to either $\min \mathcal{S}$ or zero. The former case is known as an asymmetric integer quantization (e.g., used in the Tensorflow framework), and the latter case is called symmetric quantization. The compression system presented in the paper is based on symmetric quantization, and the dynamic fixed point was also implemented. These schemes were used because they are the more generic and generally lead to the best results. Furthermore, we were not constrained with any specific platform that would require the use of a dedicated compression scheme. It is worth emphasizing that the hardware platform used for neural model implementation is a decisive factor for a compression scheme. Platforms such as FPGAs are much more flexible than DSPs or ASICs since they allow for a more aggressive compression scheme.

All of the presented approaches are examples of linear quantization; however, it is also possible to use a nonlinear version to minimize quantization loss. The hardware implementation of nonlinear quantization is more sophisticated, and it is more challenging to achieve the gain in the used hardware resources.

FPGA implementation can also give additional freedom for quantization adjustments, as it is not bound to any standard data width nor format. On a standard CPU, moving from 16-bit precision to 15-bit will not result in any performance improvements; a calculation will be executed using the same hardware. For FPGAs, however, this will result in synthesizing a smaller logic design with fewer connections, which can be beneficial for latency, throughput, or calculation density.

The order in which the quantization and pruning are applied is critical, especially for lower-precision representation. If quantization is applied first, some of the values will be removed by the pruning. With high quantization precision and a low pruning threshold, the effect is hardly noticeable. However, with an increase in the pruning threshold and a decrease in the quantization resolution, the significant portion of the available range will not be utilized. Therefore, pruning should be applied first.

3. Datasets

Our experiments are performed on the same datasets as were used in [9]; summary statistics of the datasets are provided in Table 1. Some datasets are divided into training, validation, and testing data. If the validation data is not specified, then a random 10% of the training is used for it. In the MR, Subj, CR, and MPQA datasets, the testing data was not prepared; a tenfold cross-validation was performed instead.

Table 1
Dataset statistics.

	Max words per sequence	Vocabulary size	Classes	Documents	Cross-validation	Accuracy [%] [9]
MR ¹	64	18,767	2	10,662	Yes	81.5
SST-1 ²	61	17,838	5	11,855	No	48.0
SST-2 ²	61	16,190	2	9613	No	87.2
Subj ³	128	21,324	2	10,000	Yes	93.4
TREC ⁴	45	8766	6	5952	No	93.6
CR ⁵	113	5341	2	3775	Yes	84.3
MPQA ⁶	44	6248	2	10,606	Yes	89.5

¹ Movie reviews [13], <https://www.cs.cornell.edu/people/pabo/movie-review-data/>

² Movie reviews [16], <https://nlp.stanford.edu/sentiment/>

³ Subjective/objective sentences [12], <https://www.cs.cornell.edu/people/pabo/movie-review-data/>

⁴ Question categorization [11], http://cogcomp.org/page/resource_view/8

⁵ Customer reviews [8], <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>

⁶ Opinions [17], http://mpqa.cs.pitt.edu/corpora/mpqa_corpus/

4. FPGA Implementation

4.1. Pruning

The weights in a neural network used in this work are mostly concentrated in the convolution operations (around 99% of total trained weights, with 1% in the fully connected layers). Therefore, the FPGA implementation is concentrated on reducing and accelerating the convolutional layer. As the embedding layer is not implemented in the FPGA, it is excluded from these calculations. The design was implemented and synthesized using the Xilinx Vivado HLS environment according to Alg. 1. All weights are hard-coded into the logic to allow for more-effective pruning. The loops from Lines 2-4 are fully unrolled.

The process of transforming the Keras model to an IP core that can be used in the FPGA is presented in Fig. 2. In the first step, the Keras network model is created

and trained. When the accuracy is satisfactory, the network model and weights need to be exported. A Python script processes the weights from an HDF5 file to a C++ header format. Their values must be known during compilation; otherwise, Vivado HLS will not be able to create a hard-coded network. A separate header is created for each layer. A JSON file containing the model architecture is used to create the inference function of a neural network. This function is a top-level one and performs all of the computations on the input data. At this point, we can decide how the data and weights should be represented. A simulation is performed to validate the generation process. If the results are satisfactory, Vivado HLS will synthesize the network IP core. Next, the IP core is put into the Vivado design, and the bitstream is generated.

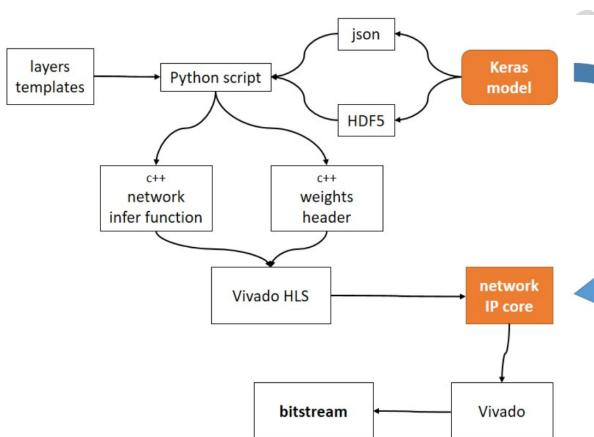


Figure 2. Processing Keras [3] model to FPGA IP core.

4.2. Quantization

The precision reduction was applied to both the weights of the layers (embeddings, two convolutional, and two dense layers) and the outputs of the activation functions (after each of the convolutional and first dense layer). The process is performed by finding a minimum and maximum of the values to be reduced and uniformly dividing the value range into 2^{bits} buckets. Each bucket has a middle point of the represented subrange assigned. Each layer weight or activation output can be quantized using a different number of bits. In the case of layer weights, the quantization is straightforward. The precision reduction of the activation functions requires running a computation on the training data and remembering the minimum and maximum of the activation functions for the chosen layers. During the testing, the computed values may fall outside this range – these need to be clipped accordingly.

5. Experiments

5.1. Uniform-precision quantization

The first set of experiments involved a precision reduction to the same number of bits for every combination of layers and activations. The tested number of bits included 32, 16, 8, 7, 6, 5, 4, 3, 2, and 1, which gives

$$2^{\text{number of layers and activations}} \cdot \text{number of tested bit-widths} = 2^8 \cdot 10 \quad (8)$$

experiments for each dataset or fold. In all of the experiments, the integer quantization resulted in a bit better accuracy than in a dynamic fixed point (difference up to 2%); therefore, it was used for the presentation of the results.

Tables 2 and 3 presents the results for the SST-2 and MR datasets with a single layer or activation quantized. In the case of 32 bits, there is no reduction, as it is default precision. Accuracy increases in some settings, and a precision reduction to 16 bits does not change the accuracy results. The embeddings can be reduced to five bits, and a majority of the dataset weights of the convolutional layers can be reduced to four bits. Usually, quantization of activations decrease the accuracy more than quantizations of the weights. Accuracy drops the most when the activations of the first dense layer are reduced to fewer than four bits. It is important to mark that the convolutional layers are parallel; reducing one of the layers to one bit does not change the overall accuracy significantly, as the second is operating with full precision.

The tables also present the results for the precision reduction of all layers and activations simultaneously (last row). It is worth noting that the quantization of all layers and activations can yield better results than when the precision is reduced in only one of them. Precision below four to five bits results in an accuracy drop of more than 1%.

Table 2

Model accuracy for **SST-2** dataset as function of precision for each layer or layer activation.

	Acc. [%] 32 bits	Accuracy change for selected precision [bits]									
		16	8	7	6	5	4	3	2	1	
embedding_1_a	85.01	0	+0.11	+0.16	+0.33	+0.11	+0.60	-0.33	-0.72	-0.83	
conv1d_1	85.01	0	0	0	-0.11	+0.05	+0.22	+0.27	0	-0.06	
conv1d_1_a	85.01	0	0	+0.05	+0.11	+0.22	+0.27	+0.44	+0.16	-0.94	
conv1d_2	85.01	0	-0.06	0	+0.05	0	+0.16	-0.11	+0.33	+0.11	
conv1d_2_a	85.01	0	-0.06	-0.06	-0.06	0	0	+0.27	-0.17	-2.31	
dense_1	85.01	0	0	-0.06	-0.11	-0.11	0	0	0	+0.16	
dense_1_a	85.01	0	+0.05	+0.05	+0.11	-0.11	-0.17	-1.59	-15.10	-32.68	
dense_2	85.01	0	-0.06	0	+0.05	+0.05	-0.06	+0.05	+0.16	-0.11	
all	85.01	0	+0.16	+0.16	+0.33	+0.16	+0.22	-1.65	-2.09	-22.30	

Table 3

Model accuracy for **MR** dataset as function of precision for each layer or layer activation.

Acc. 32 bits	Accuracy change for selected precision [bits]									
	16	8	7	6	5	4	3	2	1	
embedding_1_a	80.35	0	-0.07	-0.04	+0.01	+0.04	-0.41	-1.07	-1.58	-1.57
conv1d_1	80.35	0	+0.02	+0.02	+0.01	-0.02	-0.04	+0.05	0	-0.22
conv1d_1_a	80.35	0	0	-0.03	-0.06	-0.05	-0.04	-0.11	-0.30	-2.86
conv1d_2	80.35	0	+0.01	+0.02	-0.01	-0.01	+0.03	-0.05	-0.12	-0.45
conv1d_2_a	80.35	0	0	-0.06	-0.07	-0.06	-0.04	-0.03	-0.31	-2.55
dense_1	80.35	0	+0.01	+0.02	+0.02	-0.06	-0.05	+0.01	-0.03	+0.09
dense_1_a	80.35	0	+0.08	+0.06	-0.04	-0.28	-1.02	-4.43	-15.40	-27.49
dense_2	80.35	0	+0.01	+0.03	+0.02	-0.02	-0.03	+0.02	+0.03	-0.15
all	80.35	0	+0.04	+0.05	-0.09	-0.41	-1.51	-3.22	-3.72	-21.66

5.2. Arbitrary-precision quantization

The second experiment was focused on finding the best precision reduction with an arbitrary chosen maximal accuracy decrease. Here, we do not limit the model reduction to the same number of bits for each layer and activation. In order to test every combination (including ten values of the number of bits), it would be necessary to perform 10^8 tests (which is not computable in a reasonable amount of time). A random-restart hill-climbing algorithm was employed to tackle this problem (Alg. 2). In the beginning, the algorithm randomizes the order in which the layers and activations will be quantized. Then, it cyclically iterates through them until no further model compression is possible. For each layer, it tries to find the smallest number of bits that do not decrease the accuracy more than a set threshold. The searching method using Alg. 2 was restarted 50 times. In this work, we set the maximal accuracy decrease to 0.2% of the original.

Algorithm 2 Random-restart hill-climbing algorithm for searching maximal compression scheme.

```

threshold ← 0.998                                ▷ allow 0.2% drop from original
for all places do                                ▷ places – layers or activations undergoing quantization
    precisions[place] ← 32                      ▷ default precision of 32 bits
end for
original_accuracy ← TEST(precisions)
SHUFFLE(places)                                    ▷ shuffle order of places
while precisions is changed do
    for all places do
        n ← precisions[place]
        for precision ← 1, n do
            precisions[place] ← precision

```

```

accuracy ← TEST(precisions)
if accuracy ≥ original_accuracy · threshold then
    break
end if
end for
end for
end while

```

Table 4 shows the results of the precision reduction for each dataset. When compared to 32-bit representation, the sizes of the models can be reduced by more than 84%. However, most space is taken by the word embeddings (more than 93%). This could be further reduced by limiting the vocabulary and reducing the dimensions of the embeddings. Another approach was to order the quantized layers and activations by their influence on the model size (in this case, starting from the optimizing embedding layer). The approach did not achieve better results than with Alg. 2. The same precision bit parameters were obtained for two datasets.

Table 4

Best model-size reduction and accuracy achieved in 50 runs of Alg. 2

	Size reduction [%]	Size [MB]	Size of embeddings [%]	Accuracy [%]	
				32 bits	reduced
MR	84.69	3.419	98.14	80.35	80.21
SST-1	87.52	2.655	96.12	46.79	46.92
SST-2	93.66	1.229	94.23	85.01	84.90
Subj	87.73	3.099	98.43	92.93	92.75
TREC	88.21	1.285	97.60	90.60	91.00
CR	85.33	1.023	93.39	83.31	83.15
MPQA	85.33	1.175	95.09	89.08	88.91

The best-found precision settings are shown in Table 5. All of the precision values are equal to or fewer than eight bits.

Table 5

Best precision settings found by Alg. 2 for each dataset.

	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
embedding_1_a	5	4	2	4	4	5	5
conv1d_1	2	5	3	1	1	2	2
conv1d_1_a	5	4	2	2	4	5	7
conv1d_2	3	3	2	2	1	3	2
conv1d_2_a	4	6	3	3	2	3	6
dense_1	1	4	4	3	2	2	3
dense_1_a	7	8	5	6	4	6	6
dense_2	2	1	1	2	1	2	1

5.3. Pruning and quantization

The third experiment involved a combination of quantization and pruning; it was performed on the MR database. The data width changed from 32 bits to 1 bit, omitting the values between 32 and 16 as well as between 16 and 8 (as the changes were insignificant). Pruning was applied as described in Alg. 1. The threshold started from 0 (i.e., no pruning) to 0.14 with step size equal to 0.005. When the threshold reached 0.14, all weights were removed and no multiply and accumulate operations were performed in the convolutional layers. Figure 3 presents the extensiveness of pruning depending on a threshold. The values are calculated as a ratio of the actually performed calculations to the number of calculations that would be performed if no pruning were applied. It is visible that the ratio quickly drops; this is a result of the Gaussian distribution of weights, which are concentrated at around 0. Setting the pruning threshold to 0.035 results in around 20% of the calculations being performed, while the rest are omitted (as marked with the red dashed line in Fig. 3).

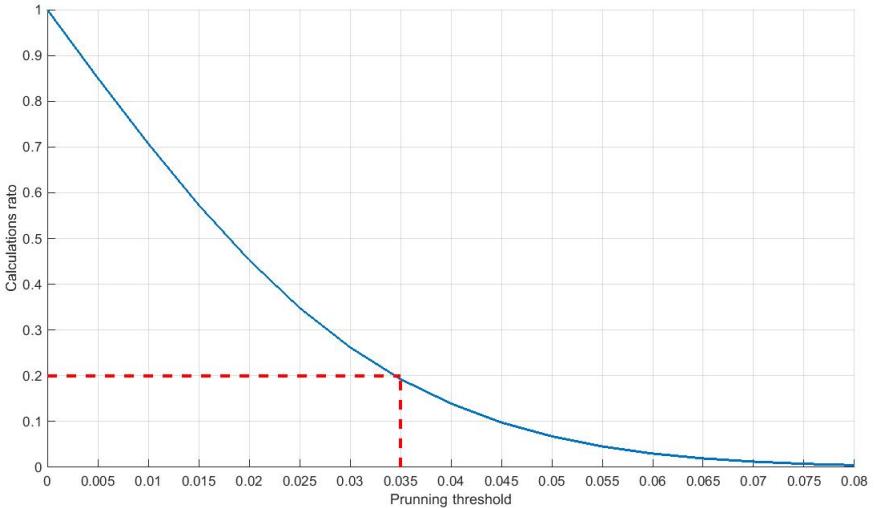


Figure 3. Ratio of calculations performed after pruning to number of calculations that would be performed without pruning.

During the experiment, we performed the network inference on all of the test samples from the MR database for each pruning threshold – quantization bit-width combination, creating a mesh of values. The impact on the accuracy of the network is presented in Fig. 4. The quantization effect on the accuracy is independent of pruning. A significant drop in accuracy appears below 3-bit precision as well as over a 0.03-0.035 pruning threshold. Between these values and no reduction starting point, a rectangular plateau (marked in semi-transparent red in Fig. 4) of only slight variations in the precision can be observed.

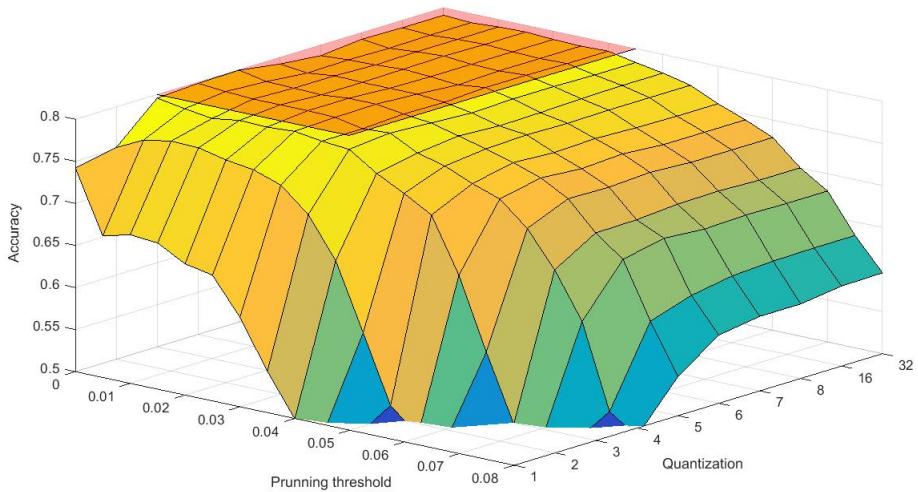


Figure 4. Accuracy results during network reduction through quantization and pruning.

5.4. FPGA results

Due to the limitations of the synthesis tools, the exact CNN-static layers could not be used. Therefore, the experimental results presented in this section are obtained from a similar but smaller version of a convolutional layer, with an input of size 64 and 35 feature maps, a filter of size 2 and output of size 63, and 16 feature maps. The impact of applying the quantization and pruning mechanisms on logic utilization, however, can be extrapolated.

We checked the pruning and quantization effects separately and in conjunction. The experiments were performed on a Xilinx Virtex-7 FPGA VC707 Evaluation Kit with a Virtex-7 series FPGA. The test points were selected so the accuracy would remain within 1% of the original. The results are summarized in Tab. 6. When both techniques are used, the flip-flop utilization drops 4.27 times and the lookup table utilization drops 4.32 times while having a minimal impact on the accuracy.

Table 6

FPGA resource utilization for selected quantization and pruning settings.

Precision	Pruning threshold	FF	LUT
32	0	8118	60,750
	0.035	4449	14,906
3	0	1952	14,553
	0.035	1899	14,047

6. Discussion

This work focuses on an intersection of CNN, NLP, sentiment analysis, and neural model compression, which may be considered as a subfield of the emerging embedded machine-learning domain. We presented the results for the model, which is somehow unique in NLP dominated by an RNN-based architecture. Our work as such is somehow complementary to [5], where the authors examine the compression schemes for natural language modeling and [15], which addresses embedding compression. It is worth noting that we also addressed RNN compression in [18]. The authors of [5] applied quantization and pruning as well as more-advanced techniques such as low-rank factorization for model compression. In [15], the multi-codebook quantization approach was applied, and the model was trained end-to-end. In our work, we proposed the random-restart hill-climbing algorithm for searching the maximal compression, which allowed us to achieve results optimal for each layer with a performance degradation of lower than 0.2 % of the original accuracy. It is worth noting that, in order to take advantage of this compression scheme, dedicated configurable hardware is essential (FPGA), which allows for adjusting the precision for each layer arbitrarily. The CPU and GPU have a fixed internal processing architecture, which prevents them from taking full advantage of the presented compression scheme. Furthermore, the proposed scheme improved the original performance of the model in some cases despite the applied compression (e.g., Table 4: TREC dataset). This also results from a different structure of the NLP data and its representation in a semantic latent space in a model where fewer more-distinct manifolds are created. In NLP concepts are more core-grained distributed. This renders the model more compression-prone, provided that an adequate algorithm is employed that is capable of spotting and extracting the manifolds. The proposed random-restart hill-climbing algorithm allows us to explore the latent space to some extent. However, to make the process more effective, an algorithm that takes the data structure into account as a prior should be used since it has a substantial impact on the performance of the models.

7. Conclusions and future work

This paper considers several compression and pruning techniques for convolutional neural networks in sentiment analysis. The results of the experiments show that the size of the model may be reduced by more than 84% with a small performance degradation of approximately 1%. Future work will concentrate on boosting the performance of the given conventional network by changing the architecture, using a deeper model, and modifying the filter scheme. Moreover, we plan to apply a memetic algorithm and reinforcement techniques to find the configuration of the network that gives the best model compression.

Acknowledgements

The research presented in this paper was partially supported by the Faculty of Computer Science, Electronics, and Telecommunications of the AGH-UST statutory tasks within the subsidy of the Ministry of Science and Higher Education.

References

- [1] Al-Hami P.K.M., Pietron M., Casas R., Hijazi S.: Towards a Stable Quantized Convolutional Neural Networks: An Embedded Perspective. In: *ICAART 2018 : 10th International Conference on Agents and Artificial Intelligence : proceedings : Funchal, Madeira, Portugal : 16-18 January, 2018.* 2018.
- [2] Cheng Y., Wang D., Zhou P., Zhang T.: Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges. In: *IEEE Signal Processing Magazine*, vol. 35(1), pp. 126–136, 2018. ISSN 1053-5888. <https://doi.org/10.1109/MSP.2017.2765695>.
- [3] Chollet F., et al.: Keras. <https://keras.io>, 2015.
- [4] Dozat T.: Incorporating Nesterov Momentum into Adam. In: *International Conference on Learning Representations 2016*. 2015. <https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ>.
- [5] Grachev A.M., Ignatov D.I., Savchenko A.V.: Neural Networks Compression for Language Modeling. In: B.U. Shankar, K. Ghosh, D.P. Mandal, S.S. Ray, D. Zhang, S.K. Pal, eds., *Pattern Recognition and Machine Intelligence*, pp. 351–357. Springer International Publishing, Cham, 2017. ISBN 978-3-319-69900-4.
- [6] Gysel P.: *Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks*. Master's thesis, University of California, 2016. <http://arxiv.org/abs/1605.06402>.
- [7] Han S., Pool J., Tran J., Dally W.: Learning both weights and connections for efficient neural network. In: *Advances in neural information processing systems*, pp. 1135–1143. 2015.
- [8] Hu M., Liu B.: Mining and Summarizing Customer Reviews. In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 168–177. ACM, New York, NY, USA, 2004. ISBN 1-58113-888-1. <https://doi.org/10.1145/1014052.1014073>.
- [9] Kim Y.: Convolutional Neural Networks for Sentence Classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751. Association for Computational Linguistics, Doha, Qatar, 2014. <https://doi.org/10.3115/v1/D14-1181>.
- [10] Laura J.A., Masi G., Argerich L.: From Imitation to Prediction, Data Compression vs Recurrent Neural Networks for Natural Language Processing, 2017. <http://arxiv.org/abs/1705.00697>.

- [11] Li X., Roth D.: Learning Question Classifiers. In: *COLING*, pp. 556–562. 2002.
- [12] Pang B., Lee L.: A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. In: *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Association for Computational Linguistics, Stroudsburg, PA, USA, 2004. <https://doi.org/10.3115/1218955.1218990>.
- [13] Pang B., Lee L.: Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 115–124. Association for Computational Linguistics, Stroudsburg, PA, USA, 2005. <https://doi.org/10.3115/1219840.1219855>.
- [14] Pennington J., Socher R., Manning C.: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. Association for Computational Linguistics, Doha, Qatar, 2014. <https://doi.org/10.3115/v1/D14-1162>.
- [15] Shu R., Nakayama H.: Compressing Word Embeddings via Deep Compositional Code Learning, 2017. <http://arxiv.org/abs/1711.01068>.
- [16] Socher R., Perelygin A., Wu J., Chuang J., Manning C.D., Ng A., Potts C.: Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, Seattle, Washington, USA, 2013. <https://www.aclweb.org/anthology/D13-1170>.
- [17] Wiebe J., Wilson T., Cardie C.: Annotating Expressions of Opinions and Emotions in Language. In: *Language Resources and Evaluation*, vol. 39(2), pp. 165–210, 2005. ISSN 1572-0218. <https://doi.org/10.1007/s10579-005-7880-9>.
- [18] Wielgosz M., Karwatowski M.: Mapping Neural Networks to FPGA-Based IoT Devices for Ultra-Low Latency Processing. In: *Sensors*, vol. 19(13), 2019. <https://doi.org/10.3390/s19132981>.
- [19] Wróbel K., Wielgosz M., Pietroń M., Duda J., Smywiński-Pohl A.: Improving text classification with vectors of reduced precision. In: *ICAART 2018: 10th International Conference on Agents and Artificial Intelligence: proceedings: Funchal, Madeira, Portugal: 16–18 January, 2018*. 2018.
- [20] Xu Y., Wang Y., Zhou A., Lin W., Xiong H.: Deep Neural Network Compression With Single and Multiple Level Quantization. In: *AAAI18 32nd Conference on Artificial Intelligence Technical Track: Machine Learning Methods*. 2018. <https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16479>.

Affiliations

Krzysztof Wróbel

Jagiellonian University, ul. Gołębia 24, 31-007 Cracow, Poland, krzysztof@wrobel.pro, ORCID ID: <https://orcid.org/0000-0002-3485-7825>

Michał Karwatowski

AGH University of Science and Technology, al. Adama Mickiewicza 30, 30-059 Krakow, Poland, mkarwat@agh.edu.pl, ORCID ID: <https://orcid.org/0000-0001-6285-136X>

Maciej Wielgosz

AGH University of Science and Technology, al. Adama Mickiewicza 30, 30-059 Krakow, Poland, wielgosz@agh.edu.pl, ORCID ID: <https://orcid.org/0000-0002-4401-2957>

Marcin Pietroń

AGH University of Science and Technology, al. Adama Mickiewicza 30, 30-059 Krakow, Poland, pietron@agh.edu.pl, ORCID ID: <https://orcid.org/0000-0001-9357-9231>

Kazimierz Wiatr

AGH University of Science and Technology, al. Adama Mickiewicza 30, 30-059 Krakow, Poland, wiatr@agh.edu.pl, ORCID ID: <https://orcid.org/0000-0001-5959-0277>

Received: 29.07.2019

Revised: 08.11.2019

Accepted: 10.11.2019