DNN Split Computing: Quantization and Run-length Coding are Enough

Damiano Carra
University of Verona, Italy
damiano.carra@univr.it

Giovanni Neglia
Inria, Université Côte d'Azur, France
giovanni.neglia@inria.fr

***Abstract*—Split computing, a recently developed paradigm, capitalizes on the computational resources of end devices to enhance the inference efficiency in machine learning (ML) applications. This approach involves the end device processing input data and transmitting intermediate results to a cloud server, which then completes the inference computation. While the main goals of split computing are to reduce latency, minimize energy consumption, and decrease data transfer overhead, minimizing data transmission time remains a challenge. Many existing strategies involve modifying the ML model architecture which ultimately requires resource-intensive retraining.**

In our work, we explore lossless and lossy techniques to encode intermediate results without modifying the ML model. Concentrating on image classification and object detection—two prevalent ML applications—we assess the advantages and limitations of each technique. Our findings indicate that simple tools, such as linear quantization and run-length encoding, already accomplish considerable information reduction, which is on par with more complex state-of-the-art techniques that necessitate model retraining. These tools are computationally efficient and do not burden the end device.

I. Introduction

In recent years, the continuous growth of computational power has facilitated the development of increasingly complex Deep Neural Network (DNN) models. The accuracy and precision achieved in various tasks, such as image classification or object detection, are approaching, if not surpassing, human performance levels [1]. However, such model complexity demands resources that may not always be available on the device executing the task of interest [2].

For example, in image classification, the device capturing the photo may be a smartphone with limited computational, memory, and energy capabilities. Consequently, the image is sent to the cloud, where dedicated servers perform the task and send the response back to the device. An alternative to this scenario, commonly referred to as *Split Computing* [3], [4], [5], involves the device executing an initial portion of the DNN and sending the intermediate result—represented as tensors—to the cloud for processing to obtain the final output.

Split computing is beneficial for several reasons, including the ability to reduce latency, energy consumption, and data transfer overhead. In particular, if the split point is chosen carefully, the amount of data sent over the network may be less than the input data (e.g., the input image size) [2]. In addition, the scheme leverages the device resources, relieving the cloud (where the resources are shared among various tasks) of some of the computation. Finally, transmitting intermediate tensors may protect input data privacy, with theoretical guarantees when differential-privacy-like techniques are used [6].

Existing literature has investigated split computing, considering the characteristics of different state-of-the-art architectures [2], [7]. The ideal split point, in terms of minimizing data transmission size, may often lie within the later layers. This unfortunately necessitates the device to shoulder a significant portion of the computational load. Some studies have then proposed introducing a bottleneck for encoding the intermediate tensors of the initial layers to reduce the data transmission [4], [5]. However, this solution necessitates to carefully design a suited bottleneck and then retrain the new model. Moreover, most studies have assumed that tensor coefficients are transmitted as is [2], [8], overlooking whether the statistical characteristics of these coefficients render them suitable for effective compression or quantization. Only a few papers have noted that moderate levels of quantization do not impact the final accuracy [4], [9].

In this work, we systematically examine various lossless and lossy techniques for encoding the intermediate tensor of a DNN and their potential impact on final accuracy. Our focus is on solutions that do not require any modifications to the DNN architecture; instead, they can be applied to pre-trained, out-of-the-box, state-of-the-art architectures and remain viable if model parameters are updated to prevent progressive quality deterioration [10].

We consider well-established techniques such as quantization, differential coding, and run-length coding, which are prevalent in different areas of information communication or representation. We also explore other techniques, like the identification of the most significant coefficients through Grad-CAM, which are specific to the field under consideration. Although these techniques have been used to investigate the inner workings of DNNs, they can also be employed to explore potential data compression. To our knowledge, no previous study has analyzed them collectively, assessing their impact on the amount of data transmitted and the accuracy of the task to be accomplished.

We consider two tasks: image classification and object detection for video sequences. In image classification, each instance of the task can be treated as independent (e.g., the device needs to classify a single picture), while object detection for videos allow us to examine potential temporal correlations among tensors related to consecutive frames.

Our results reveal that extreme quantization—using merely three bits per coefficient—effectively reduces the data to be transmitted marginally affecting the accuracy of the final output. Moreover, the abundance of zeros allows for run-length encoding, further decreasing the data size. These low-complexity techniques—requiring only two passes on the tensor to produce the coded data and one pass for decoding—achieve a compression ratio comparable to that of more complex bottleneck approaches, which necessitate retraining the DNN. At the same time, we find that there is minimal or no spatial or temporal correlation within and between tensors, limiting the potential benefits of compression techniques that rely on such correlations.

The remainder of the paper is organized as follows. In Section II, we provide background information and discuss related work. In Section III, we investigate the spatial and temporal correlation of the intermediate tensors, while in Section IV, we explore Grad-CAM. We discuss quantization in Section V and our coding approach in Section VI. Finally, we conclude the paper in Section VII.

II. Background and Related Work

We consider two machine learning (ML) applications, namely, image classification and object detection, in which tasks are completed with a single forward pass. This allows computation to be resumed at any layer using the input tensor for that layer. For object detection, the DNN (such as YOLO [11]) possess a more intricate structure, involving the concatenation of different layers. Despite this complexity, the first part of the architecture consists of a sequence of layers that can be utilized as potential split points.

Split computing has garnered significant attention in recent years, with a growing number of works in this area. In [7], the authors present a survey on recent developments. Here, we discuss some representative studies to highlight the limitations of previous works.

Both Neurosurgeon [2] and JointDNN [8] explore the optimal split point by taking into account the total delay, which encompasses both processing time and transmission time. Their findings indicate that the last layers of the DNN are the most suitable for splitting. However, such a split would require the device to carry out the majority of the work, and as DNN architectures grow increasingly complex, this approach might become impractical. For instance, in [2], the optimal delay for the simpler AlexNet [12] was under 0.1 s, but it rose to nearly 1 s for VGG16 [13], mainly due to increased device processing latency.

To minimize the inference delay on the device, it is crucial to target splitting the DNN at one of its initial layers. In image classification tasks, two widely used DNN architectures are VGG16, with 30 layers, and ResNet-50 [14], with 22 layers—although most of the layers consist of blocks containing other layers, such as convolutional and ReLU—along with its evolution, ResNet-152 (56 layers). Table I shows the transmitted tensor sizes for the two architectures under consideration at the optimal splitting point, (the point that minimizes this size). Sizes are expressed in bytes, assuming each coefficient is represented by a 4-byte float. The table also includes the average size of JPEG images in ImageNet validation dataset (50000 images), which is commonly used to evaluate DNN model accuracy. It is important to note that ImageNet images generally require some pre-processing to be adapted to the model, which expects a 3-channel 224x224 px image as input. As a result, in practice the actual size of the input tensor is on average more than 4.5 times larger than the original JPEG image. Furthermore, the tensor size at the split points is 6–12 times the size of the input image.

TABLE I
Classification: Size of the JPEG input (average considering the ImageNet dataset), of the input tensor and of some internal tensors for different DNN architectures.

		Ratio w.r.t.	
	Size (kB)	**JPEG**	**input tensor**
Image (JPEG)	128.5	1.0	0.22
Input tensor	588.0	4.57	1.0
VGG16, layer 10	1568.0	12.20	2.67
VGG16, layer 17	784.0	6.10	1.33
ResNet-50, layer 4	784.0	6.10	1.33
ResNet-50, layer 8	1568.0	12.20	2.67
ResNet-152, layer 16	784.0	6.10	1.33
ResNet-152, bottleneck [4]	10.6	0.082	0.014

Due to the impracticality of transmitting tensors of such large sizes, the authors of [15] explore model quantization, which involves quantizing model weights and tensor coefficients. While this approach does reduce the amount of data to be transmitted, the use of quantization for both weights and coefficients substantially affects accuracy. In contrast, our work focuses on tensor quantization without altering the model, demonstrating that it does not negatively impact accuracy. Other studies [16], [17] consider lossy compression for transmitted data but necessitate DNN retraining to minimize accuracy loss. Furthermore, these studies, along with the ones focused on tensor coding such as [18], do not discuss the complexity of the compression scheme. Our solution quantizes and encodes the tensor exclusively with minimal computational complexity. By scanning the tensor coefficients only twice, the process is remarkably fast, and its impact becomes negligible when compared to the transmission time of the JPEG image or the total inference time.

Another group of studies involves modifying the DNN [4], [19], [20], [9]. Specifically, they introduce additional layers, such as an Autoencoder, to create a *bottleneck*—a layer in the network with minimal number of neurons—and split the DNN at the bottleneck. This approach, commonly known as *bottleneck injection*, achieves favorable compression ratios and has a computational cost at inference time comparable to the original DNN. For instance, the last row of Table I presents the encoded data size for ResNet-152 with a bottleneck at layer 16 (results taken from Table 4 in [4]). This method significantly reduces the amount of data to be transmitted, while maintaining limited impact on accuracy. All these solutions based on bottlenck injection share a common characteristic: they require retraining a portion of the DNN, which poses a burden every time a model is updated. In contrast, our strategy leaves the DNN unaltered; it solely processes the tensor at the split point while still achieving, as demonstrated in Section VI, the same compression rate as the bottleneck approach.

The above observations apply to object detection as well. Table II presents the average image size for the COCO dataset [21] and the tensor size at the optimal splitting point for the YOLOv5 architecture, which is currently the state-of-the-art for this task. In [9], the authors introduce a bottleneck in Faster R-CNN (the state-of-the-art at that time) resulting in a transmitted tensor 0.64 times smaller than the input image [9, Table IV]) and 0.066 times smaller than the smallest tensor in the original architecture. As shown in the next sections, also in this case, our solution obtains a comparable compression rate with no modification to the DNN.

TABLE II
OBJECT DETECTION: SIZE OF THE JPEG INPUT (AVERAGE CONSIDERING THE COCO DATASET), AND OF THE INTERNAL TENSOR.

		Ratio w.r.t.	
	Size (kB)	**JPEG**	**split tensor**
Image (JPEG)	159.1	1.0	-
Yolov5m, layer 4.m	1800.0	11.31	1.0
Bottleneck (Faster R-CNN) [9]	101.8	0.64	0.066

III. Spatial and temporal correlation

We begin our analysis by investigating the potential to compress the transmitted tensor by utilizing spatial and/or temporal correlation. Specifically, we focus on the tensor generated by a single image or frame at the split point. Unless otherwise specified, we consider the split points with the fewest coefficients for each architecture—layer 17 for VGG and layer 4 for ResNet-50—although the same observations apply to other split points and architectural variants, such as ResNet-152.

Figure 1 shows the empirical CDF of the tensor coefficients—we present results for four sample inputs, but the same observations apply to any input. For VGG and ResNet-50, the ReLU activation function produces many zeros, comprising between 40% and 50% of the coefficients, while the remaining coefficients have values within a limited range (less than 20 for VGG and less than 2 for ResNet). In the case of YOLOv5 (Fig. 3, left), where the leaky ReLU is utilized, a significant proportion of coefficients fall below zero, but they have a limited range (between -1 and 0). In all cases, the restricted range justifies the exploration of quantized representations that may save bits during transmission (Section V).

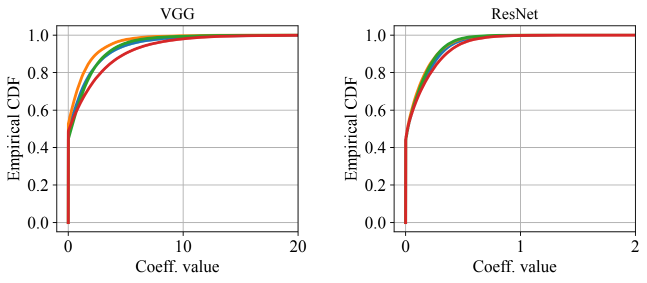

Fig. 1. Empirical CDF of the tensor coefficients for two DNN architectures. Each figure contains the CDF derived from 4 randomly chosen input images from the ImageNet dataset.

Spatial correlation. We examine the spatial correlation across the three dimensions of the tensor (row, column, and channel) by differentially encoding consecutive coefficients within the same row, column, or channel, which involves subtracting the previous coefficient from the current one. If neighboring coefficients have similar values, the distribution of the differentially encoded coefficients should display a significantly smaller range, with a high concentration of values around zero.

Figure 2 depicts the distribution when differential encoding is applied across rows (similar observations apply when encoding over columns or channels). We can see that the distribution support becomes more compact, with the range being approximately halved for some images. However, this reduction in support does not necessarily result in significant savings when encoding the information. For instance, if we maintain the same quantization error, a halved range would only save a single bit per quantized coefficient.

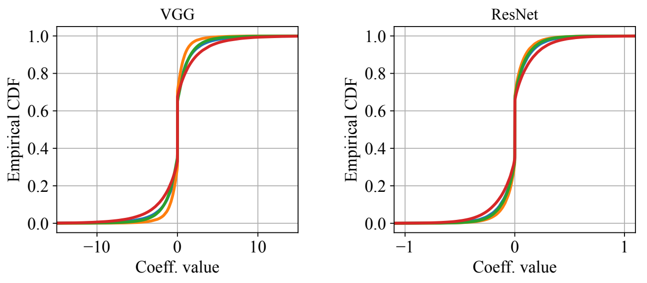

Fig. 2. Empirical CDF of the of differentially encoded tensor coefficients.

In the final step, we investigate spatial correlation in the frequency domain as well. To achieve this, we vectorize the tensor using various orders for its three dimensions, and then apply a fast Fourier transform (FFT) to the resulting vector. Transmitting only the lower-frequency coefficients results in a substantial loss of inference accuracy (specific results are not presented here due to space constraints). This further indicates that there is no significant statistical regularity in the spatial distribution of the coefficients that can be leveraged to decrease the amount of transmitted information.

Temporal correlation. In the context of object detection for video streams, two consecutive frames typically differ by only a subset of pixels, given that the majority of the scene tends to remain static. When applying split computing, it is reasonable to expect that the intermediate tensors associated with two consecutive frames might have some common coefficients. As a result, applying differential encoding in these situations should produce a more compact representation.

We conducted tests on various videos featuring nearly static scenes, such as a person sitting in a chair who then stands up and walks away. The chair remains at the same location and is easily detectable, and the background does not contain any objects of interest. Even in this simplified scenario, the range of differentially encoded coefficients and uncoded ones differ only by roughly a factor two, as illustrated in Figure 3.

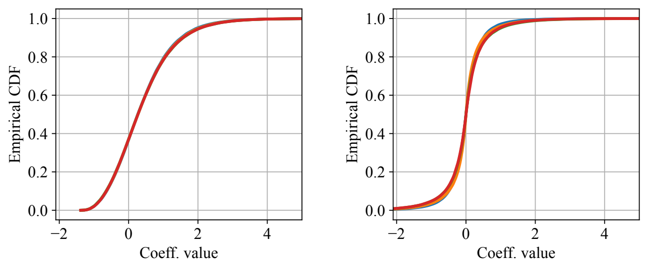

Fig. 3. Empirical CDF of the the tensor coefficients (left) and differentially encoded tensor coefficients (right) for a set of consecutive inputs (YOLO architecture).

Conclusion. Although spatial and temporal correlation has proven effective in coding images and videos, for split computing, the tensor coefficients do not exhibit a significant dependence that can be leveraged to reduce the amount of information transmitted. However, the distribution of coefficients does possess a limited range and includes numerous zeros, which can be directly utilized to achieve this goal.

IV. Coefficient relevance: Grad-CAM

In the previous section, we noted that at the examined split point, a substantial portion (more than 40%) of the tensor coefficients are zeros. This observation raises the question of whether positive coefficients play a more crucial role in the task. If they do, it would be beneficial to retain as much of their information as possible during lossy processing.

To answer this question, we adopt a saliency-based approach [22]. Saliency-based approaches were originally proposed to evaluate the relevance of input features x by calculating the partial derivative of the softmax value y^c for the input's predicted class c, i.e., $\partial y^c / \partial x$. As the derivative reveals how much a change in each input feature would affect the classification output, we can expect that the most relevant features are indeed those that contribute the most to the classification. A specific approach to identify relevant image pixels, Grad-CAM, was proposed in [23]. Examples in [23] show that, when training an image classifier to distinguish cats and dogs, the relevant portion of the image is indeed that with the cat or the dog, while the background and other objects in the scene are ignored. Grad-CAM was extended in [19] to identify which DNN layers have the highest impact on the final result.

In our case, we evaluate the relevance of each tensor coefficient at the split point, by applying the Grad-CAM approach at the split point, instead of the last convolutional layer.

Results. We consider the 10% most relevant coefficients as identified by Grad-CAM. Figure 4 compares the distribution of values for the most relevant coefficients with the distribution of all coefficients. Although the relevant coefficients are more likely to have larger values than non-relevant ones, we find that there are relevant coefficients with very small values. Specifically, for ResNet, Grad-CAM identifies most of the null coefficients as relevant. We observed the same behavior at different split points and with other architectures (e.g., ResNet-152), as well as for different input images belonging to various classes.

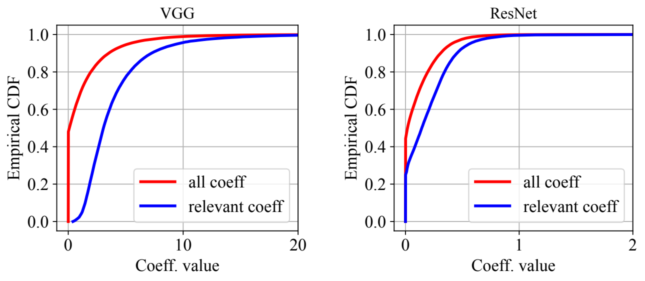

Fig. 4. Empirical CDF of the coefficients for two DNN architectures, and comparison with the relevant coefficients identified by the saliency analysis.

Conclusion. The saliency approach can be employed to pinpoint the tensor coefficients that have the most significant impact on the task output. However, these coefficients cannot be easily distinguished based solely on their values.

V. Quantization

Quantization is a lossy form of compression, which inevitably affects the final model performance (e.g., in terms of accuracy or precision). To mitigate such an impact, some studies on quantized DNNs suggest re-training the DNN by incorporating the effect of quantization into the loss function [16]. Since our goal is to work with off-the-shelf architectures, we focus on quantizing the coefficients to be transmitted, which are then restored as 4-byte floats on the cloud side before continuing with DNN processing. While some research [4] has investigated the effects of quantization in split computing, it has been limited to bottlenecked DNN architectures and has considered at least 8 bits. In contrast, we apply quantization to the split layer tensor coefficients and explore extreme (linear) quantization, using as few as 2 bits per coefficient.

Figure 5 demonstrates the effect of quantization on the classification task's accuracy (left) for various architectures. For all architectures except ResNet-50, using 4 bits per coefficient does not significantly impact the accuracy compared to the architecture without quantization. By removing an additional quantization bit, image accuracy is still comparable to that obtained by the bottleneck injection approach [5]. For the object detection application, which is more complex and involves estimating the bounding boxes of detected objects, we present two precision metrics, mAP50 and mAP50-95, which are typically provided when evaluating a model. A 4-bit quantization has no impact on the metric (Fig. 5, right). A 3-bit one suffers a slight loss in precision, which is comparable to the loss experienced by bottlenecked architectures [9].

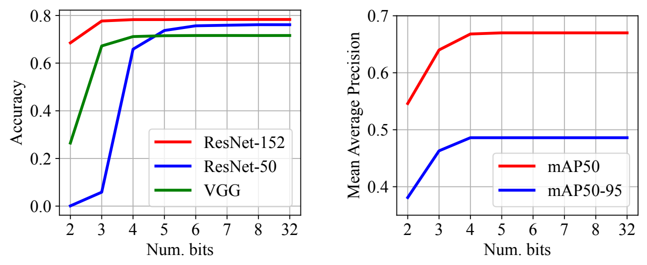

Fig. 5. Impact of the number of quantization bits on the classification accuracy (left) and on the detection mean average precision (right).

Since quantization narrows down the set of possible coefficient values, it might enhance their spatial or temporal correlation. For this reason, we also investigated the potential for differential encoding of quantized coefficients, but this approach yielded limited advantages, similarly to what observed in Section III. On the other hand, in the next section, we will see that some specific form of spatial correlation (the long sequences of zeros) can be beneficially exploited.

Conclusion. Quantization of the tensor coefficients to be transmitted, up to 3 bits per coefficient, has minimal or no impact on the accuracy or precision of the task performed by the DNN. This observation holds true across various tested architectures (VGG, ResNet-152, YOLOv5) and tasks (classification, detection).

VI. TENSOR CODING

Quantization enables a more compact representation of tensor coefficients. We now investigate the possibility of achieving even more compact encoding. Previous research has employed various schemes, such as conventional image or video codecs [16] or entropy coding [17]. However, these methods demand high computational resources, resulting in additional coding delays. For example, compressing a tensor using the zlib library [24] takes an average of 25.2 ms [1] which is approximately 1/4 of the transmission time for a raw JPEG image over a stable 10 Mbps channel.

We notice that, due to the presence of ReLU or Leaky ReLU, after quantization, we often see long sequences of the minimum value of the quantization interval, which is represented by zero. In contrast, other quantized values typically do not appear in sequences. For this reason, we employ a modified version of run-length encoding, where zeros are coded with run-length while other coefficients remain unchanged. The first codeword bit indicates if what follows is the binary representation of the coefficient ($b = 0$) or the number of zeros in the sequence minus 1 ($b = 1$). For example, the 3-bit quantized sequence:

```
3, 4, 0, 0, 0, 0, 0, 7
```

is translated into (spaces between coefficients added for clarity)

```
0011 0100 1100 0111
```

Note that in this way we can use 4-bit codewords to represent sequences of up to 8 zeros. In case of sequences of zeros longer than 8 (*e.g.*, 20), we code consecutive zero sequences (*e.g.*, 8, 8, 4), so the decoder needs always to process 4 bits at a time. The extension to n-bit codewords is immediate.

From a computational standpoint, the quantization and coding can be performed with two passes of the tensor. In the first pass, we identify the minimum and maximum values. In the second pass, each coefficient is quantized (using min-max linear quantization) and coded in a single step. The minimum and maximum values of the coefficients are placed as 4-byte values at the beginning of the data to be sent. The decoding can be done in a single pass: once the minimum and maximum values have been read, each codeword can be read and immediately translated into the corresponding tensor coefficient. Note that the quantization scheme may easily be made adaptive by specifying as metadata the number of bits used for quantization.

Results. We applied our proposed tensor coding method to the output of selected layers of the different architectures, and compute the average size of the data to be transmitted over the reference datasets, ImageNet (for classification) and COCO (for detection). Figure 6 shows the result for different numbers of quantization bits together with the average size of the JPEG image in the corresponding dataset. The compression ratios achieved through simple quantization and run-length coding fall within a similar range as those obtained by bottleneck DNNs in [4], [5], which require retraining the model. Notice that our coding is lossless and any impact on the accuracy is solely due to quantization and has been evaluated in Section V.

Comparison with standard compression. We compare our tensor coding method with general-purpose compression techniques like zlib [24], which can be applied to either the original tensor or its quantized version. In the first case, we observed that the transmitted tensor retains about 60% of its original size, and then is nearly 4 times larger than the JPEG input image. This option, besides being computationally expensive, is not of practical interest. In the second case, significant size reduction is achievable. Nevertheless, Figure 7 (left) shows that our simple scheme achieves even larger savings, but for

[1]Average computed over the ImageNet dataset, using ResNet-50 architecture, tensor at layer 4.

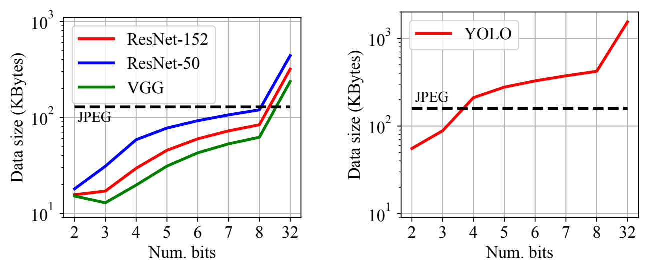

Fig. 6. Impact of the number of quantization bits on the size of the data to be transmitted for classification (left) and detection (right). The black dashed line indicates the average size of the input JPEG image.

extreme values of quantization (less than 4 bits). Moreover, our scheme shines in terms of computational cost. The quantization requires two tensor passes and run-length encoding can be performed during the second pass essentially with no additional cost. On the contrary, zlib requires additional computation after the tensor is quantized. Figure 7 (right) shows that, on an Intel i9-10900X CPU 3.70GHz, zlib may require between two and three times more time if ResNet-50 is split after its 4th layer. Results for other architectures and layers lead to similar conclusions.

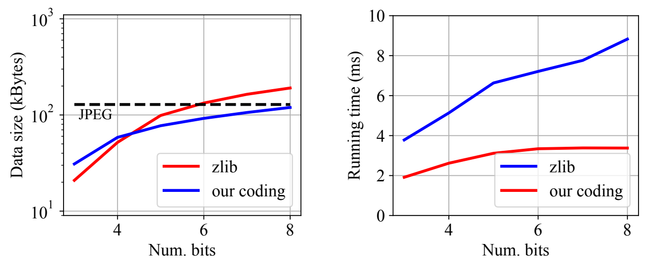

Fig. 7. Data size obtained using zlib compared to our scheme (left) and running time for achievening it (left), with ResNet-50 architecture. The black dashed line indicates the average size of the input JPEG image.

Conclusion. The abundance of zeros among the tensor coefficients allows for a straightforward and efficient coding method that, in conjunction with quantization, can represent the information within the tensor in a compact manner. This approach has minimal or no impact on accuracy and does not require any modification to the DNN architecture.

VII. Conclusion

The goal of split computing is to leverage the computational resources of a device and reduce the load on cloud servers, which are increasingly being used for various services. Achieving this is beneficial if data transmission does not significantly impact the overall delay. Consequently, we explored different compression techniques to address this challenge.

For two widely adopted machine learning applications, classification and detection, we analyzed the advantages and limitations of various mechanisms. Our focus was on processing the tensor to be transmitted without considering any architectural modification. Such modifications require updates whenever the reference architecture changes, for instance, due to re-training with new data.

Our findings revealed that the internal DNN tensors do not exhibit a significant level of spatial or temporal correlation, limiting the potential benefits of compression techniques relying on such correlations. However, simple tools like linear quantization and run-length encoding are sufficient to achieve a considerable compression ratio.

As future work, we plan to investigate the effect of transmission loss on final accuracy. This could provide insights into the use of reliable or unreliable communication channels.

References

[1] S. Dodge *et al.*, “Human and dnn classification performance on images with quality distortions: A comparative study,” *ACM Transactions on Applied Perception (TAP)*, vol. 16, no. 2, pp. 1–17, 2019.
[2] Y. Kang *et al.*, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.
[3] A. E. Eshratifar *et al.*, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *IEEE/ACM ISLPED*, 2019, pp. 1–6.
[4] Y. Matsubara *et al.*, “Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems,” *IEEE Access*, vol. 8, pp. 212 177–212 193, 2020.
[5] M. Sbai *et al.*, “Cut, distil and encode (CDE): Split cloud-edge deep inference,” in *IEEE SECON*. IEEE, 2021, pp. 1–9.
[6] J. Wang *et al.*, “Not just privacy: Improving performance of private deep learning in mobile cloud,” in *ACM SIGKDD*, 2018, pp. 2407–2416.
[7] Y. Matsubara *et al.*, “Split computing and early exiting for deep learning applications: Survey and research challenges,” *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.
[8] A. E. Eshratifar *et al.*, “JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services,” *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.
[9] Y. Matsubara *et al.*, “Neural compression and filtering for edge-assisted real-time object detection in challenged networks,” in *IEEE ICPR*, 2021, pp. 2272–2279.
[10] D. Vela *et al.*, “Temporal quality degradation in ai models,” *Scientific Reports*, vol. 12, no. 1, p. 11654, 2022.
[11] J. Redmon *et al.*, “You only look once: Unified, real-time object detection,” in *IEEE CVPR*, 2016, pp. 779–788.
[12] A. Krizhevsky *et al.*, “Imagenet classification with deep convolutional neural networks,” *Comm. of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[13] K. Simonyan *et al.*, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
[14] K. He *et al.*, “Deep residual learning for image recognition,” in *IEEE CVPR*, 2016, pp. 770–778.
[15] G. Li *et al.*, “Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge,” in *ICANN*. Springer, 2018, pp. 402–411.
[16] H. Choi *et al.*, “Deep feature compression for collaborative object detection,” in *IEEE ICIP*, 2018, pp. 3743–3747.
[17] R. A. Cohen *et al.*, “Lightweight compression of neural network feature tensors for collaborative intelligence,” in *IEEE ICME*, 2020, pp. 1–6.
[18] S. R. Alvar *et al.*, “Pareto-optimal bit allocation for collaborative intelligence,” *IEEE Transactions on Image Processing*, vol. 30, pp. 3348–3361, 2021.
[19] F. Cunico *et al.*, “I-split: Deep network interpretability for split computing,” in *IEEE ICPR*, 2022, pp. 2575–2581.
[20] G. Castellano *et al.*, “Regularized bottleneck with early labeling,” in *ITC 2022-34th International Teletraffic Congress*, 2022.
[21] T.-Y. Lin *et al.*, “Microsoft coco: Common objects in context,” in *ECCV*. Springer, 2014, pp. 740–755.
[22] J. Adebayo *et al.*, “Sanity checks for saliency maps,” *Advances in neural information processing systems*, vol. 31, 2018.
[23] R. R. Selvaraju *et al.*, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *ICCV*, 2017, pp. 618–626.
[24] zlib, https://www.zlib.net/, Accessed: 2023-04-15.