Early Exiting with Compressible Activations for Efficient Neural Network Inference

Jona Beysens[1], Martin Sénéclauze[1], Philippe Dallemagne[1], Siavash Bigdeli[2]
[1]Swiss Center for Electronics and Microtechnology (CSEM), Switzerland
[2]Technical University of Denmark (DTU), Denmark
Email: {jona.beysens, martin.seneclauze, philippe.dallemagne}@csem.ch, sarbi@dtu.dk

***Abstract*—Cloud-based Machine Learning (ML) incurs high latency and high sensitivity to connectivity failures. This can be improved by maximizing the ML processing on the local device. However, since these low-power devices typically have limited resources, the early-exit mechanism has been used to hierarchically split a neural network in parts over multiple devices, trading off computation with communication costs. But, the intermediate activations can be significantly large at the split point. In this paper, we present a novel entropy-based technique that learns to intelligently compress activations during training and efficiently encodes them during inference. We show that in an early-exit configuration, entropy regularization with Huffman coding can save up to 54% in communication cost, while keeping the classification accuracy of MobileNetV2 on CIFAR-10 above 85%.**

***Index Terms*—Hierarchical machine learning, activation compression, entropy regularization, early exiting**

I. Introduction

Traditional cloud-based Machine Learning (ML) inference typically exhibits high latency, power-hungry communication and low robustness [1]. Moreover, privacy is at risk in cloud-based solutions. This can be improved by processing as much as possible locally on the device, ideally without the need for the cloud [2]. However, in practice the cloud is still required due to the lack of computational resources on the local device (i.e., available processing power and memory). To get the best of two worlds, the ML model can be split in parts between the local end device and the cloud.

A straightforward choice is to split the model and transfer the intermediate activations among the end device and cloud. However, state-of-the-art Convolutional Neural Networks (CNNs) have large intermediate activations, especially at their first layers. Fig. 1 presents the raw (i.e., unencoded) activation size in INT8 format of state-of-the-art CNNs on the CIFAR-10 dataset, confirming that intermediate activations are significantly larger than the input size itself for state-of-the-art image classification models (e.g., $48\times$ for MobileNetV2).

A more efficient approach is to split the model and use early exiting [3], [4]. This technique adds an extra exit to one of the hidden layers of an ML model. Easy samples will be processed locally by the early exit on the end device, while difficult samples will be handled by the final exit on the cloud.

This paper is a preliminary version of the work that is accepted for publication in IEEE Smart Systems Integration 2023.

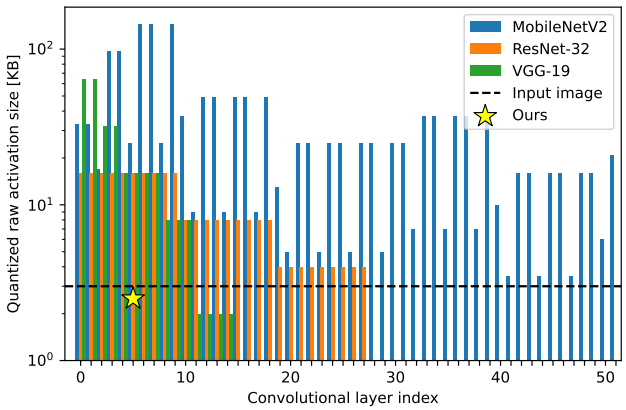

Fig. 1. Activation size of state-of-the-art CNNs on CIFAR-10 dataset. 'Ours' is our entropy-based compression for MobileNetV2 (cfr. Table IV, $\lambda_e = 1.0$).

The latter leads to added energy consumption and latency, as difficult samples need to be sent over the network.

In this work, we propose a novel solution that learns to compress intermediate activations for efficient communication between end device and cloud on image classification tasks. The learning happens during training, without the need to adapt the ML architecture, nor it requires extra processing during inference. We show that in an early-exit configuration, entropy regularization with Huffman coding can save up to 54% in communication cost, while keeping the classification accuracy of MobileNetV2 on CIFAR-10 above 85%.

II. Related work

The related work on activation compression is extensive and involves different techniques. Pruning and quantization are widely studied to reduce the complexity of ML models [5], [6]. The work in [4] uses and additional compression stage (i.e., loss-less LZ4 algorithm) to further reduce the activation size at the split point. In contrast to traditional compression, other works introduce an extra ML model specifically designed to reduce the data size over the network [7], [8]. DynO proposes a dynamic algorithm adapting the partition point and data precision to the execution environment [9]. Activations can also be compressed through sparsity regularization, which is extensively studied in literature [10], [11]. However, as shown in Sec. V-D, this only works at sparsity-inducing layers (e.g., ReLU layer). Deep compression models are often applied to video and image data by directly compressing the

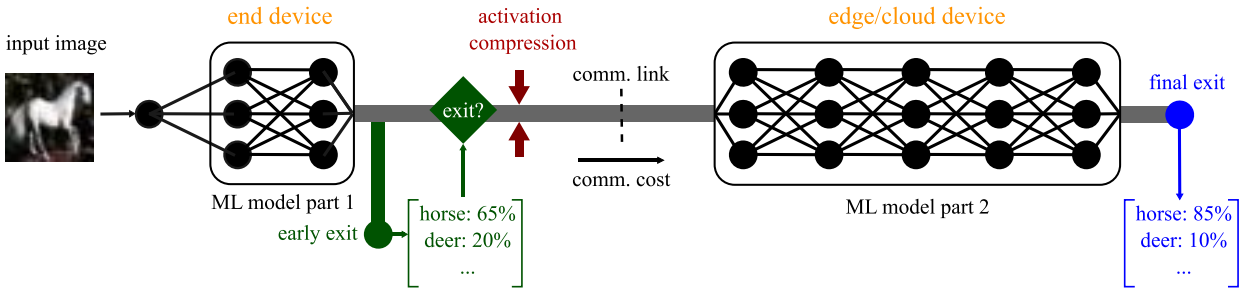

Fig. 2. System architecture, visualizing the three pillars considered in this work: 1) model splitting, 2) early exiting, and 3) activation compression.

neural network activations [12]–[14] in an end-to-end fashion. Density-based compression models use explicit probabilities to compress their input data [15]–[17].

We propose instead a novel regularization technique featuring an all-in-one model for classification and compression. It learns how to compress activations in the design phase, minimizing the overhead at inference. Further, it works on a wide set of layers, including no sparsity-inducing layers.

III. System Model

We demonstrate our approach for an image classification task. Fig. 2 presents the overall system architecture, consisting of three pillars: 1) model splitting, 2) early exiting, and 3) activation compression. The ML model is split into N_s parts. Without loss of generality, we use $N_s = 2$ in this paper; the model is split between the end device and edge/cloud device.

Before every split point, an early exit is added to the original model. This early exit will decide whether the intermediate activations at the split point needs to be transferred to the edge/cloud device. As such, the ML model has $N_s - 1$ early exits and one final exit at the end.

We assume that the training is performed exclusively in the cloud, whereas the inference is split across multiple devices. At inference, activations are transferred over the network, which incurs a communication cost. We define the communication cost as the activation size (encoded or unencoded) at the split point that needs to be sent over the network.

To avoid confusion between a neural network and a communication network, in the remainder of this work, we use the terminology ML model for the former and network for the latter. Further, we refer to the feature maps at the hidden layers of the ML model as the *activations*.

A. Model splitting

In principle, a CNN model can be split at any hidden layer. The design parameters for selecting the split point include amongst others, the available memory on the end device, the communication cost, and expected robustness in case of connectivity outages. For instance, for an ML model with residual connections, it is favorable to split the model after the residual connection, to avoid the need to exchange the extra residual branch over the network, which would otherwise double the communication cost. The choice of its position will be explained in more detail in Sec. V.

B. Early exiting

We use the early-exit mechanism to reduce the processing power and communication cost during inference [3], [4]. We place an early exit just before every split point, to fully exploit the available memory on the end device.

An early exit consists of 1) an average pooling layer with a kernel size of three and a stride of two to reduce the data size, and 2) a fully connected layer as output. It is followed by a softmax layer to normalize the output into a list of class probabilities. These represent the confidence level at which a class belongs to the input image. If the maximum confidence level exceeds the predefined confidence threshold $T_c \in [0; 1]$, the early exit is confident enough about its result and no further processing is required. If not, the calculations continue until the final exit, which is then responsible for the classification. The confidence threshold T_c is a parameter set by the user. It tunes the ratio of samples handled by the early exit versus the ratio of samples communicated over the network and processed by the final exit. To train the ML model with early exits, we use the following objective loss function L_o:

$$L_o = \sum_{i=0}^{N_e-1} \rho_i \, L_{ce,i}, \tag{1}$$

in which $N_e = N_s$ represents the total number of exits (i.e., number of early exits plus the final exit), $L_{ce,i}$ the cross-entropy loss at exit i and ρ_i its weighting coefficient. In this work, we use $\rho_i = 1/N_e$, to pay equal attention to all exits.

C. Activation quantization and encoding

After training the ML model in floating point format (FP32), the weights and activations are quantized to integers (INT8). At inference time, the quantized activations are encoded into a bit stream. We consider two loss-less encoding techniques: 1) Huffman coding and 2) Compressed Sparse Row (CSR) coding. They will be compared with the raw activation size, in which all INT8 values are encoded with eight bits.

Huffman coding requires to build a codebook before encoding, mapping all symbols into binary codewords. We construct a codebook for every split of the ML model, such that the most probable symbol (activation) is encoded by the fewest amount of bits, minimizing the overall codeword size [18]. The codebook can be efficiently stored on the end device using canonical codes, which require to save only the bit-lengths [19]. It takes a memory overhead of $\mathcal{O}(n)$, with n the

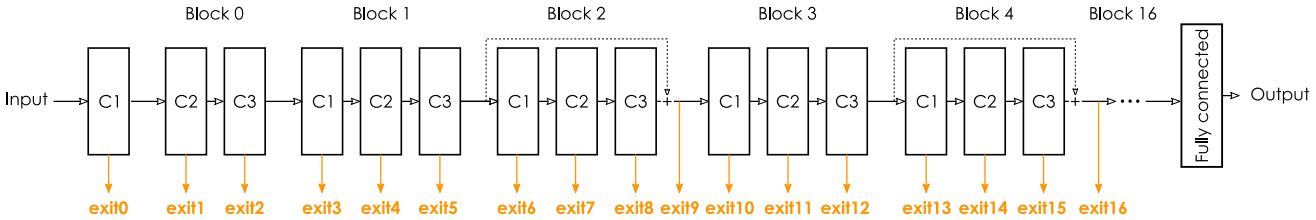

Fig. 3. Block diagram of MobileNetV2, with early exits represented in orange.

number of codewords. As such, encoding INT8 integers yields a limited overhead of maximally 256 B on the end device.

In CSR coding, only non-zero values and their indices are encoded [20]. Suppose we need to encode a matrix of size $P \times Q$ with R non-zero elements. By encoding the values themselves (raw encoding), you need to store PQ bytes. With CSR coding, you need to store $2(P+1)+3R$ bytes. As such, it does not depend on the number of rows Q. For sparse matrices with a low value of $R \ll Q$, CSR coding is favorable.

IV. Activation Compression

To compress activations, we employ regularization during training. This technique modifies the loss function to favor smaller activations. As such, it requires no additional compression stages, nor needs to adapt the ML model architecture. The total loss function L_t is now given by:

$$L_t = L_o + L_r, \tag{2}$$

with L_o the objective loss (cfr. Sec. III-B) and L_r the regularization loss. L_t should be differentiable, as the backpropagation algorithm needs gradient information during training.

The L_r loss exclusively regularizes the activations of the hidden layer at the split point, which need to be communicated over the network. These are usually the activations that are fed into the early exit branch. We do this to minimize the cost (drop in accuracy), while maximizing the gain (reduction in communication cost). To achieve this, we investigate two regularization techniques: 1) reducing activation entropy and 2) increasing activation sparsity, which are discussed next.

A. Entropy-based regularization

Lower activation entropy results in a smaller encoded data size. To accomplish this, we consider the following entropy-based regularization loss $L_{r,e}$:

$$L_{r,e} = \frac{\lambda_e}{\log_2 N_b} \left[- \sum_{j=0}^{N_b-1} h_j \, \log_2 h_j \right], \tag{3}$$

with N_b the number of bins in the estimated activation histogram H, h_j the value of the j^{th} histogram bin, and λ_e the regularization weighting coefficient which controls the attention paid to the entropy level. As such, we approximate the activation probability distribution by its histogram in every training batch. We normalize the entropy level to its maximal value, which is determined by N_b. Without loss of generality, we use $N_b = 256$ in this work, matching it with the size of the quantized activations after training.

To estimate the histogram, we first remove outliers in the activations if the samples are outside the range $[\mu-3\sigma; \mu+3\sigma]$, where μ and σ are the running mean and standard deviation of the activations over the training batches. Then, we normalize the activations to the range $[0; 1]$. To preserve gradient information during backpropagation, we estimate the histogram using a triangular weighting function [21]. We consider an histogram H having N_b bins, with nodes $t_1 = 0$, t_2, ..., $t_{N_b} = 1$ uniformly filling the range $[0; 1]$ and step size $\Delta = 1/N_b$. The value h_j of the histogram H is then estimated as:

$$h_j = \frac{1}{N_a} \sum_{k=0}^{N_a-1} \delta_{j,k}, \tag{4}$$

in which N_a represents the number of activation samples in the current training batch and x_k the k^{th} normalized activation sample. The weights $\delta_{j,k}$ are chosen such that each activation sample is assigned to its two adjacent nodes:

$$\delta_{j,k} = \begin{cases} (x_k - t_{j-1})/\Delta, & \text{if } x_k \in [t_{j-1}; t_j], \\ (t_{j+1} - x_k)/\Delta, & \text{if } x_k \in [t_j; t_{j+1}], \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

B. Sparsity-based regularization

Higher activation sparsity (i.e., fraction of zero elements) also reduces the encoded data size. We evaluate sparsity regularization as a baseline in our comparison. To induce this, we consider the following L1-norm regularization loss $L_{r,s}$:

$$L_{r,s} = \frac{\lambda_s}{N_a} \sum_{k=0}^{N_a-1} |x_k|, \tag{6}$$

with $|x_k|$ the absolute value of the k^{th} activation sample, and λ_s the regularization weighting coefficient controlling the level of attention paid to the sparsity level.

V. Performance Evaluation

This section presents the evaluation results of the early exit mechanism and activation compression techniques.

A. Setup

We evaluate our work on an image classification task using the CNN MobileNetV2 with width multiplier 1.0 [22], ResNet-32 [23] and VGG-19 [24] on the CIFAR-10 dataset [25]. Fig. 3 depicts the block diagram of MobileNetV2. The model is structured in 17 blocks. Each block consists of three layers: an expansion layer extending the activations to a higher-dimensional space (referred to as C1), a depth-wise

TABLE I
MOBILENETV2 BLOCK STRUCTURE.

Sub-block type	Notation	Modules
Expansion	C1	2D conv. + batch norm + ReLU
Depth-wise conv.	C2	2D conv. + batch norm + ReLU
Compression	C3	2D conv. + batch norm

Fig. 4. Performance evaluation workflow.

convolution layer performing the separable convolutions (C2) and a compression layer bringing the activations back to a lower-dimensional subspace (C3). Table I shows the structure of the layers. A subset of blocks have a residual connection, bypassing all layers inside the block (e.g., block 2 and 4 in Fig. 3), to mitigate the vanishing gradient problem [26].

We consider multiple split points with corresponding early-exit positions, as depicted in Fig. 3. They are placed at the end of every sub-block; after the ReLU layer for C1 and C2, while after the batch norm layer for C3. Fig. 4 presents the performance evaluation worfklow. We first train the MobileNetV2 vanilla model without any early exit from scratch during $E_{base} = 300$ epochs. The model with the best top-1 accuracy on the validation set is retained. Fig. 5 shows the activation size to be communicated over the network, if we would attach an early exit at that position. We observe that the compression layers (exit 2, 5, 8, etc.) are the smallest. We conclude that it is best to split the model at these positions.

Our software implementation is based on Distiller, an open-source Python package developed by Intel for neural network compression [27]. We extended it to access fine-grained activation information and support various early-exit and regularization configurations.

B. Early exiting

We take the pre-trained vanilla model, add a single early exit to the model and continue training for an additional $E_{exit} = 50$ epochs. We keep the model with the best overall validation top-1 accuracy, which is calculated as the average accuracy over the early and final exit. We repeat this process for every early exit. As such, we obtain a new ML model for each considered early exit.

Fig. 6 shows the overall top-1 test accuracy versus the confidence threshold $T_c \in [0; 1]$. $T_c = 0$ represents the scenario in which all samples are processed by the early exit, while at $T_c = 1$ the final exit handles all samples. We observe that the higher the confidence threshold, the higher the accuracy; more samples will be processed by the final exit, hence the better performance.

Attaching an exit to a later layer shows a better initial accuracy (84.9% at exit 15) than an early layer (68.7% at exit 2), leading to a smaller jump in accuracy from $T_c = 0$ to

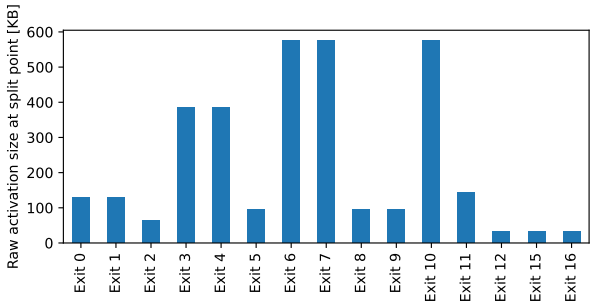

Fig. 5. Activation size of MobileNetV2 in FP32 format at split points.

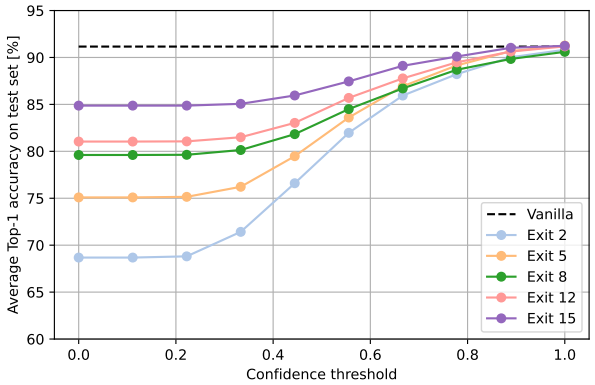

Fig. 6. Accuracy of MobileNetV2 versus T_c, which are represented by the dots. The accuracy of the vanilla network is shown by the dashed line. The further the early-exit position, the better the accuracy.

$T_c = 1$. Further, the earlier the exit, the sooner the accuracy increases from left to right in the figure. This is because for a given confidence threshold, the rate at which samples exit at the split is lower for an earlier exit. For the sake of simplicity, we do not report the configurations with exits after the residual connection (exit 9 and 16), as they unanimously perform worse than the exits just before (exit 8 and 15).

Table II presents the number of weights to store the model, as well as the number of multiply-accumulate (MAC) operations to process inference of a single input. We observe that the early-exit branch can have a high impact on the weights at the end device. However, the impact on the MACs is small, as the exit branch consists of a dense layer, which as opposed to a convolutional layer, requires relatively few MACs.

Fig. 7 shows the relation between the accuracy and the communication cost to process an image at inference time. The communication cost is presented for the raw activation size (i.e., not considering Huffman or other encoding techniques). By tuning T_c (represented by the dots), early exiting provides

TABLE II
EARLY-EXIT RESOURCE PARTITIONING FOR MOBILENETV2 IN FP32. VALUES FOR THE EARLY-EXIT BRANCH ALONE ARE SHOWN IN BRACKETS.

Configuration	End device		Edge/cloud device	
	Weights [KB]	Ops. [M MAC]	Weights [KB]	Ops. [M MAC]
Exit 2	147.1 (140.6)	1.7 (0.04)	8'597.3	86.3
Exit 5	235.8 (210.9)	6.6 (0.05)	8'578.9	81.5
Exit 8	268.9 (210.9)	15.0 (0.05)	8'546.8	73.1
Exit 12	154.8 (61.3)	20.0 (0.02)	8'510.3	68.0
Exit 15	209.5 (61.3)	23.6 (0.02)	8'455.5	64.4

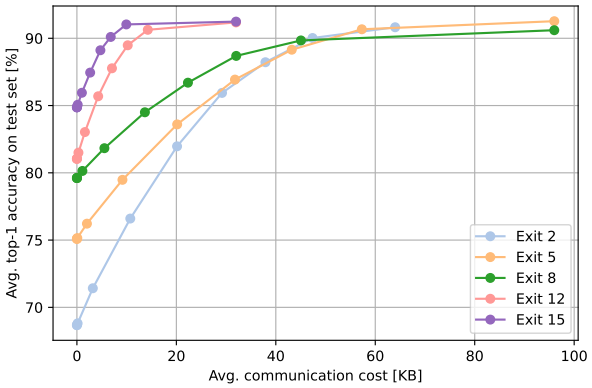

Fig. 7. Accuracy of MobileNetV2 versus communication cost at inference time. The dots represent different confidence thresholds T_c. The lower T_c, the more samples exit early and thus the less communication cost.

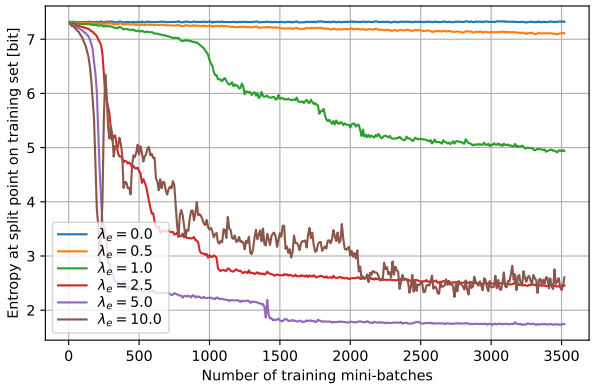

Fig. 8. Entropy of activations at split point after early exit 5 versus training mini-batches, for different regularization weights λ_e on MobileNetV2.

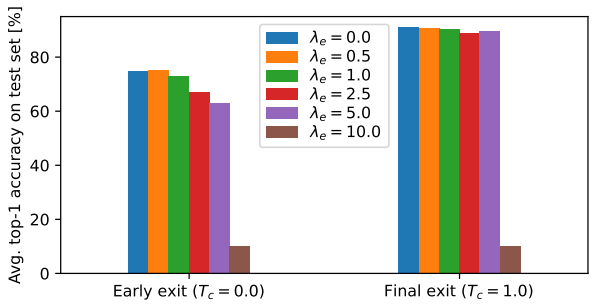

Fig. 9. Impact of entropy regularization on top-1 accuracy, for different regularization weights λ_e on MobileNetV2 with early exit 5.

a trade-off between communication cost and accuracy. For instance, by increasing the confidence threshold from $T_c = 1.0$ to $T_c = 0.9$, the overhead reduces from 32.0 KB to 9.6 KB, while loosing only 0.1% points in accuracy.

Selecting the position of an early exit in practice depends on the available memory, communication cost and MAC count on the end device. For instance, with a lower number of MACs, the end device can go faster into sleep mode, decreasing its overall energy consumption.

C. Entropy-based regularization

In this section, we investigate the impact of entropy-based regularization. We take the trained early-exit models from the previous section and continue training for an additional $E_{reg} = 10$ epochs, while adding Eq. (3) to the loss function. Without loss of generality, we select the configuration with early exit 5 to guide us through our analysis. Exit 5 is attached to sub-block C3 (cfr. Fig. 3). As such, we regularize the output of the batch norm layer, i.e., the last layer in this sub-block.

Fig. 8 shows how the activation entropy level evolves during training for different regularization weights λ_e. We observe an entropy level of 7.3 bit at the start, which is close to its maximal level (cfr. Sec. IV-A). We note that the larger λ_e, the lower entropy level. However, Fig. 9 shows that this can have a negative impact on the accuracy. We note that $\lambda_e = 1.0$ strikes a good balance; the entropy drops by 33% to 4.9 bit, at the cost of an accuracy drop of only 1.9% points at $T_c = 0$.

For vanilla Resnet-32 on CIFAR-10 with entropy regularization, we observe a similar trend; $\lambda_e = 1.0$ reduces the entropy after the ReLU layer of the second block from 6.8 to 1.1 bit, while the accuracy drops slightly from 87.6% to 86.8%. For VGG-19 on CIFAR-10, $\lambda_e = 1.0$ lowers the entropy after the max pooling layer of the second block from 2.6 to 0.6 bit, while the accuracy reduces from 88.1% to 85.0%.

D. Activation compression

1) Quantization: To optimize the required computational resources at interference time, we first reduce the precision of the weights and activations by Post-training Quantization (PTQ) using PyTorch's fbgemm backend [28]. We calibrate the quantization engine using 25% of the validation set. After quantizing the configuration with early exit 5, the early and final exit accuracy drop just by 0.70% and 0.34% points on average over all regularization weights, respectively.

2) Regularization: We first perform sparsity regularization for early exit 5 (cfr. Sec. IV-B). However, since the corresponding split point contains a batch norm layer, it yields a negligible sparsity level below 1%, making it unsuitable for this configuration. Entropy regularization shows better results. Table III shows the impact of λ_e on the activation size and the accuracy of the quantized model. We note that by reducing the precision to INT8, the raw activation size reduces $4\times$ to 24 KB (neglecting bias and scale overhead), compared to the max communication cost in Fig. 7. Huffman coding can reduce the activation size up to 76%. We note that even without regularization, Huffman coding optimizes the activation size by 40%. Due to the negligible sparsity level at the split point, CSR coding leads to a higher activation size for all

TABLE III
IMPACT OF ENTROPY REGULARIZATION ON TEST SET FOR EARLY EXIT 5.

Regularization configuration	Activation size at split [KB]			Top-1 accuracy at early exit [%]	Top-1 accuracy at final exit [%]
	Raw	CSR	Huff.		
$\lambda_e = 0.0$	24.0	66.5	14.5	74.8	91.2
$\lambda_e = 0.5$	24.0	60.4	11.4	74.6	90.6
$\lambda_e = 1.0$	24.0	31.3	8.1	71.5	89.6
$\lambda_e = 2.5$	24.0	68.8	5.8	66.2	88.8
$\lambda_e = 5.0$	24.0	70.9	5.6	61.7	89.3
$\lambda_e = 10.0$	24.0	72.0	6.3	10.0	10.0

TABLE IV
IMPACT OF ENTROPY REGULARIZATION ON TEST SET FOR EARLY EXIT 12.

Regularization configuration	Activation size at split [KB]			Top-1 accuracy at early exit [%]	Top-1 accuracy at final exit [%]
	Raw	CSR	Huff.		
$\lambda_e = 0.0$	8.0	22.5	4.8	80.2	90.8
$\lambda_e = 0.5$	8.0	19.4	3.7	80.8	90.7
$\lambda_e = 1.0$	8.0	9.4	2.5	78.3	89.5
$\lambda_e = 2.5$	8.0	3.6	1.5	64.7	62.8
$\lambda_e = 5.0$	8.0	0.1	1.0	16.0	10.0

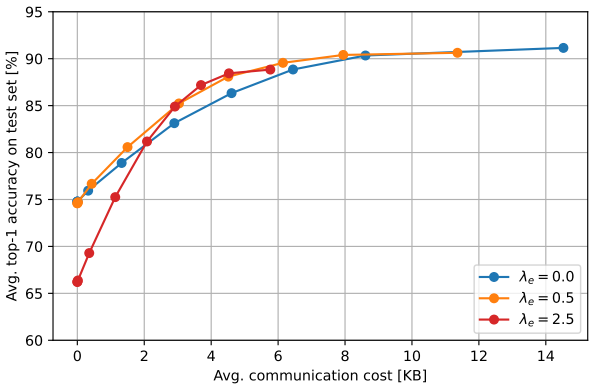

Fig. 10. Impact of entropy regularization on accuracy for configuration with early exit 5 and Huffman coding, after quantization. The dots represent different T_c. Entropy regularization reduces the average communication cost.

TABLE V
IMPACT OF SPARSITY REGULARIZATION ON TEST SET FOR EARLY EXIT 4.

Regularization configuration	Activation size at split [KB]			Top-1 accuracy at early exit [%]	Top-1 accuracy at final exit [%]
	Raw	CSR	Huff.		
$\lambda_s = 0.0$	96.0	64.7	19.3	75.4	90.4
$\lambda_s = 1e-5$	96.0	50.8	17.1	75.3	90.7
$\lambda_s = 1e-4$	96.0	17.6	13.7	73.2	89.2
$\lambda_s = 1e-3$	96.0	3.6	12.3	66.1	58.6
$\lambda_s = 1e-2$	96.0	0.1	12.0	28.9	14.3

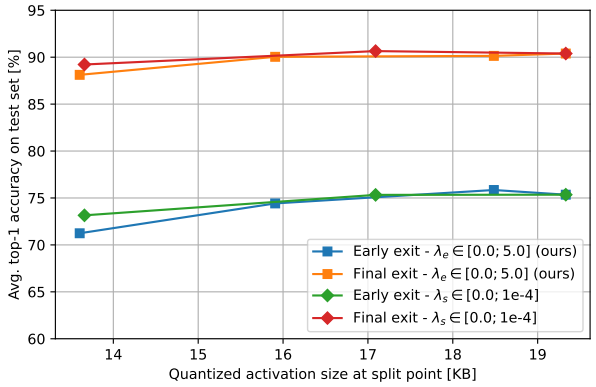

Fig. 11. Comparison of entropy and sparsity regularization for configuration with early exit 4 and Huffman coding, after quantization. The markers represent different λ_e and λ_s. The two techniques obtain similar results.

λ_e. Table IV lists the result for early exit 12, showing that we can reduce the activation size to 2.5 KB without significant accuracy loss, making it even smaller than the input image.

3) communication cost: We analyze the impact of configurations $\lambda_e = 0.5$ and $\lambda_e = 2.0$ on the average communication cost. Fig. 10 shows the results, in which the dots represent different confidence thresholds. By regularizing, we can push the graph to the left, reducing the communication cost while minimizing the accuracy loss. Suppose the application demands a minimal accuracy of 85%. Without regularization, Huffman coding can lower the communication cost from 6.5 KB to 3.9 KB. Regularizing entropy with $\lambda_e = 0.5$, can reduce this further to 3.0 KB, an additional decrease of 25%. In total, we can reduce the communication cost by 54%.

E. Comparison entropy and sparsity-based compression

To have a fair comparison between entropy and sparsity regularization, we also analyze the configuration with early exit 4, which has a ReLU layer at its split point. Table V shows the result for sparsity regularization. We observe that CSR now outperforms the raw size, but the winner is Huffman coding for all reasonable configurations (in which the final exit accuracy is higher than the early exit accuracy). Fig. 11 compares entropy and sparsity regularization for this early exit using Huffman coding, showing that both regularization techniques obtain similar performance.

Hence entropy regularization yield similar performance as sparsity regularization after a sparsity-inducing layer. However, entropy regularization provides a more widely applicable technique as it can be added to any layer of the ML model.

F. Discussions

We showed an efficient way to compress and communicate ML model's activations by adding an entropy loss to the early exit layer. We note that alternatively one could transfer a compressed input image instead of compressed activations. However, this would require an additional compression stage on the end device at inference. Further, the edge/cloud device would need to repeat calculations done on the end device, wasting computational resources. In addition, an effective compression algorithm might not be easily available for all data types (e.g., LIDAR images, thermal images, radar data [29]). Finally, privacy is at risk, as the input data might contain privacy-sensitive information. Although the input could be partly restored based on intermediate activations as well [30], recent techniques [31] show promising results for enhancing the privacy, with which our work is fully compatible.

For future work, we aim to prototype our model on low-power embedded devices, showing the gain in latency and energy consumption. We will also investigate a more general approach to automatically find the optimal early exit and compression layers.

VI. CONCLUSION

In this work, we showed that splitting an ML model in parts can result in large intermediate activations, leading to a high communication cost. To overcome this, we proposed a novel technique using early exiting and entropy-based activation compression. We demonstrated that in an early-exit configuration, entropy regularization with Huffman coding can save up to 54% in communication cost, while keeping the classification accuracy of MobileNetV2 on CIFAR-10 above

85%. This paves the way towards efficient hierarchical neural network inference.

REFERENCES

[1] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge ai: Algorithms and systems," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2167–2191, 2020.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[3] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[4] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2020.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2016.

[6] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[7] "Efficient Privacy Preserving Edge Intelligent Computing Framework for Image Classification in IoT," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, 2022.

[8] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "JALAD: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018, pp. 671–678.

[9] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 6, 2022.

[10] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, B. Nell, N. Shavit, and D. Alistarh, "Inducing and exploiting activation sparsity for fast neural network inference," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[11] G. Georgiadis, "Accelerating convolutional neural networks via activation map compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7085–7095.

[12] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, "Dvc: An end-to-end deep video compression framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 006–11 015.

[13] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Conditional probability models for deep image compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4394–4402.

[14] D. Mishra, S. K. Singh, and R. K. Singh, "Deep architectures for image compression: a critical review," *Signal Processing*, vol. 191, p. 108346, 2022.

[15] S. A. Bigdeli, G. Lin, T. Portenier, L. A. Dunbar, and M. Zwicker, "Learning generative models using denoising density estimators," *arXiv*, 2020.

[16] D. Kingma, T. Salimans, B. Poole, and J. Ho, "Variational diffusion models," *Advances in neural information processing systems*, vol. 34, pp. 21 696–21 707, 2021.

[17] Y. Song, C. Durkan, I. Murray, and S. Ermon, "Maximum likelihood training of score-based diffusion models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1415–1428, 2021.

[18] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[19] E. S. Schwartz and B. Kallick, "Generating a canonical prefix encoding," *Commun. ACM*, vol. 7, no. 3, p. 166–169, 1964.

[20] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics (SIAM), 2011.

[21] E. Ustinova and V. Lempitsky, "Learning deep embeddings with histogram loss," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2016, p. 4177–4185.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2015.

[25] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[26] A. Veit, M. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2016, p. 550–558.

[27] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural network distiller: A python package for dnn compression research," October 2019.

[28] D. S. Khudia, J. Huang, P. Basu, S. Deng, H. Liu, J. Park, and M. Smelyanskiy, "FBGEMM: enabling high-performance low-precision deep learning inference," *arXiv*, vol. abs/2101.05615, 2021.

[29] S.-P. Lee, N. P. Kini, W.-H. Peng, C.-W. Ma, and J.-N. Hwang, "Hupr: A benchmark for human pose estimation using millimeter wave radar," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023, pp. 5715–5724.

[30] A. Dosovitskiy and T. Brox, "Inverting visual representations with convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4829–4837.

[31] Y. Sepehri, P. Pad, C. Kündig, P. Frossard, and L. A. Dunbar, "Privacy-preserving image acquisition for neural vision systems," *IEEE Transactions on Multimedia*, pp. 1–12, 2022.