Reichman University
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Dynamic-Deep Compression: Tuning Cloud Costs and ECG Task Performance

by

Elad Wasserstein

M.Sc. dissertation, submitted in partial fulfillment of the requirements
for the M.Sc. degree, research track, School of Computer Science
Reichman University (The Interdisciplinary Center, Herzeliya)

March, 2022

This work was carried out under the supervision of **Dr. Eli Brosh & Prof. Anat Bremler-Barr** from the Efi Arazi School of Computer Science, Reichman University.

Acknowledgment

Several people accompanied me in the journey toward accomplishing this thesis. First, Dr Eli Brosh my direct supervisor, who was there in every junction making sure we pursue interesting insights worth spreading. Throughout his assistance I have added to my toolbox methodologies that will help me in future research. Thank you Eli for spending so many hours improving my writings and making sure my thesis's work is coherent.

A huge thanks to Prof. Anat Bremler-Barr who helped me initiate this journey. From ideating to defining an interesting research question and to the ability of placing any resource needed. Your help allowed me to move fast and concentrate on the important parts of the thesis.

Special thanks to Elad Levy for his valuable feedback regarding the architecture design. Having an experienced ML researcher around helped me finalize the architecture and make it tight.

Also, thank you Guy Vinograd from Bio-T for your insightful comments, and for bringing a customer perspective. This helped me make a research not only for the sake of research but also for solving a real-world problem the industry faces.

Finally, thank you to my wife for providing full support as well as happy distractions to rest my mind outside of my research, and to my baby girl for sleeping calmly on my chest while I am writing these words.

Abstract

Monitoring medical data, e.g., Electrocardiogram (ECG) signals, is a common application of Internet of Things (IoT) devices. Compression methods are often applied on the massive amounts of sensor data generated prior to sending it to the Cloud to reduce the storage and delivery costs. A lossy compression provides high compression gain (CG), but may reduce the performance of an ECG application (downstream task) due to information loss. Previous works on ECG monitoring focus either on optimizing the signal reconstruction or the task's performance. Instead, we advocate a lossy compression solution that allows configuring a desired performance level on the downstream tasks while maintaining an optimized CG that reduces Cloud costs.

We propose Dynamic-Deep, a task-aware compression geared for Cloud systems. Our compressor is trained to optimize the CG while maintaining the performance requirement of the downstream tasks chosen out of a wide range. In deployment, the IoT edge device adapts the compression and sends an optimized representation for each data segment, accounting for the downstream task's desired performance without relying on feedback from the Cloud.

We conduct an extensive evaluation of our approach on common ECG datasets using two popular ECG applications, which includes heart rate (HR) arrhythmia classification. We demonstrate that Dynamic-Deep can be configured to improve HR classification F1-score by a factor of 3 and increases CG by up to 83% compared to the previous state-of-the-art (autoencoder-based) compressor. Analyzing Dynamic-Deep on the Google Cloud Platform, we observe a 97% reduction in cloud costs compared to a no compression solution.

To the best of our knowledge, Dynamic-Deep is the first proposal to focus on balancing the need for high performance of cloud-based downstream tasks and the desire to achieve optimized compression in IoT ECG monitoring settings.

Contents

1 Introduction **4**

2 Related Work **8**

3 Motivation For Balancing CG And Downstream Tasks Performance **10**

4 Dynamic-Deep High Level Design **12**

5 Dynamic-Deep Implementation Details **12**

5.1 Multiple Compression Levels 12

5.2 Minimizing Memory Footprint For IoT Support 13

5.3 Combined Loss Function 14

5.4 Training 15

5.5 Tuning Loss Weighting Factors 15

6 Experimental Results **16**

6.1 Datasets and Downstream Tasks 16

6.2 Task Awareness Evaluation 17

6.3 Tuning Compression Level And Compression Gain 18

6.4 Low Correlation Between Downstream Tasks and Reconstruction Error 20

6.5 Cloud Cost Reduction Analysis Using Dynamic-Deep 21

7 Conclusion **22**

Appendices **26**

A Evaluate Multiple Downstream tasks **26**

B Dynamic-Deep Low Level implementation details **26**

List of Figures

1 The P, Q, R, S, T points are unique points that helps characterize a single interval in ECG signals. The time and value relations between the points is used to diagnosing heart disease [21] . . . 5

2 Typical architecture of a modern IoT-based medical monitoring system with data compression. . . . 5

4 Configuring an upper bound error enable a dynamic compression approach . . . 8

5 The loss's quartiles over test ECG data segments (of size 602 as in [19]) for various tasks. . . . 11

6 Dynamic-Deep shared layers reduce memory footprint . . . 14

7 Benchmark on HR arrhythmia classification task shows Dynamic-Deep successfully balance between desired performance and CG . . . 18

8 Benchmark on RPNet (R-R extraction) shows Dynamic-Deep successfully balance between desired performance and CG . . . 18

9 Theoretical feedback-aware comparison with different compression level for the HR arrhythmia classification task . . . 20

10 Compare different feedback-awareness strategies using HR arrhythmia classification loss . . . 21

11 Yearly cost expanses compression comparison . . . 22

12 Benchmark on multiple downstream tasks shows Dynamic-Deep successfully balance between desired performance and CG . . . 26

13 High Level Modules . . . 27

14 Shared encoder transforming input signal into 2-compression level . . . 28

15 Adapting different compression level to a single shared decoder (decoder_32_64_shared_iot), see Fig. 16 . . . 29

16 Shared decoder reconstruct the signal back . . . 30

17 *dense layers* implementation that learns to predict downstream task loss . . . 30

List of Tables

1 Comparison of compression methods on 602 test data segments using the CinC datasets when applied on HR arrhythmia classification task and reconstruction task. 4

2 Compression Module's Memory minimization 14

Table 1: Comparison of compression methods on 602 test data segments using the CinC datasets when applied on HR arrhythmia classification task and reconstruction task.

	CG	Avg. signal reconstruction error [%]	HR classification F1-score (Precision,Recall)	Violation of upper bound** [%]
LZ77 (Lossless)	2.7	0	**0.87*** (0.87,0.87)	**0**
CAE (SOTA)	32.25	**2.73**	0.20 (0.32,0.15)	10.21
Dynamic-Deep #1	48.31	3.81	0.73 (0.72,0.74)	**0**
Dynamic-Deep #2	**52.2**	4.2	0.69 (0.698,0.697)	**0**

* The HR arrhythmia classification F1-score (Precision,Recall) results when applied to uncompressed data segments

** Percentage of data segments that experience classification error above 0.75

1 Introduction

Internet of Things (IoT) devices are widely used to monitor and send sensor data to the Cloud for centralized storage and execution of downstream tasks. For example, hospitals use IoT medical devices to constantly monitor Electrocardiogram (ECG) signals, the patient heart's activity over time. ECG signal is a critical tool for diagnosing heart disease, and its analysis can alert the staff of abnormal heart behavior [19]. For diagnostic heart disease purposes, typical downstream tasks extract key-points in ECG intervals like R-R peaks [23] or QRS complexes [15] (illustration of those key-points in Fig. 1). Such medical monitoring settings generate large amounts of continuous sensor data to be sent to the Cloud. Transmitting the raw signal would imply power-hungry devices and high processing and storage Cloud costs. Therefore, an effective data compression scheme is required to reduce the transmission and storage requirements. An efficient compression module is typically deployed on the device to accommodate settings with low power resource-limited IoT devices, while a decompression module is deployed in the Cloud. The compression gain (CG) metric is used to evaluate such data compression scheme by calculating the division sizes of the original representation against the compressed representation. Fig. 2 presents an end-to-end data flow in a Cloud-based monitoring system[1]: compressed data segments are prepared for transmission, and the data is decompressed back to raw ECG signal in the Cloud for further processing and analysis.

General ECG compression techniques belong to one of two categories: lossless or lossy. Lossless compression methods preserve the signal's complete information but tend to achieve low compression gain (CG). In contrast, lossy techniques (that are a natural fit to IoT settings) obtain high CG at

[1]Some systems place additional servers on-site to gateway and aggregate multiple sensors. In such settings, compression is applied at the servers.

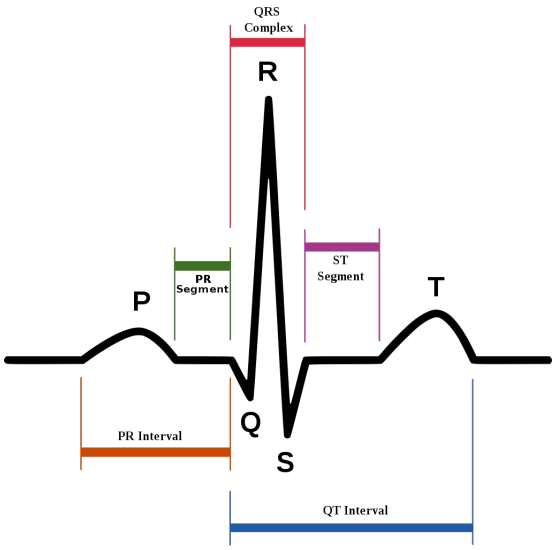


Figure 1: The P, Q, R, S, T points are unique points that helps characterize a single interval in ECG signals. The time and value relations between the points is used to diagnosing heart disease [21]

the expense of losing information, and hence reduce the performance of downstream tasks. For example, we show in Table 1 that a common lossless compression, LZ77 [24], achieves a CG of 2.7 (with STD of 0.08) on widely used ECG datasets with 602 test data segments of 2K samples each (see full datasets specs in Section 6.1), rendering the scheme inappropriate in many scenarios. On the other hand, utilizing a state-of-the-art (SOTA) lossy compression scheme, based on a convolutional autoencoder (CAE) [26] tuned to a high fixed CG of 32, leads to significant performance degradation in signal reconstruction and heart rate classification when evaluating on the CinC test dataset.

In Cloud settings, it is desired to have a lossy compression scheme sensitive to Cloud costs that maintains a high performance for one or more downstream tasks executed on the decompressed data.

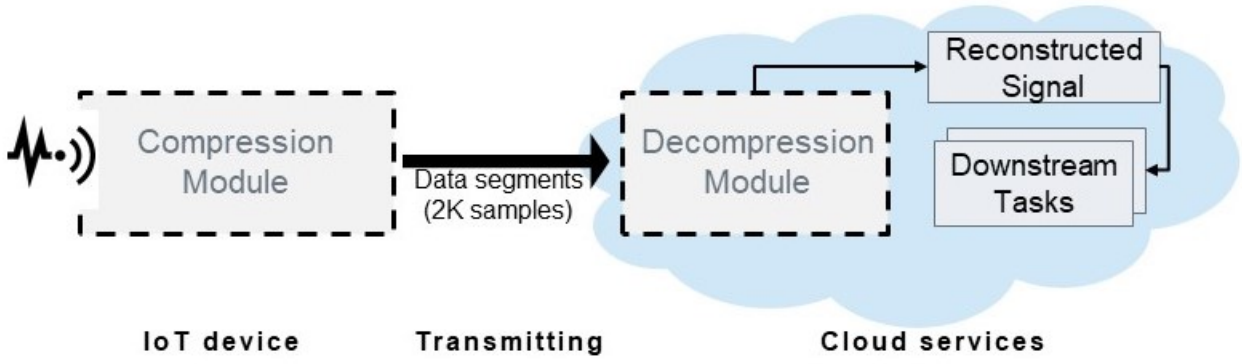


Figure 2: Typical architecture of a modern IoT-based medical monitoring system with data compression.

Previous ECG works focus either on compressing schemes optimized for signal reconstruction (and can result in low downstream task performance) or models optimized for extracting or classifying ECG features (and not necessarily lend themselves to effective compression) [15]. One approach to bridge the gap between the two is to systematically control the tradeoff between compression level and tasks' performance.

In this paper, we aim to provide a lossy compression method that allows the system administrator (admin) to configure a desired performance level, namely an upper bound on downstream tasks' error, while maintaining an optimized CG. Moreover, the method needs to be resource efficient to fit in an IoT device without adding significant overhead.

We propose Dynamic-Deep (see Fig. 3), a task-aware variable-rate compression. Our analysis reveals a low correlation between reconstruction error and the (tested) downstream task's error. The low correlation may be caused by the objectives difference, continuously tracking the signal vs. capturing sporadic events. Hence, Dynamic-Deep jointly optimizes CG and the tasks' performance level. It learns to choose the optimal CG dynamically for each data segment sent to the Cloud out of multiple compression levels, each implemented by a single convolutional autoencoder. The dynamic behavior allows the scheme to achieve impressive CG. With such a design, one can directly specify the desired performance level of the tasks, rather than provide a general hyper-parameter to control a rate quality tradeoff, as often done in variable rate compression schemes.

A straightforward approach to extend an existing CAE to support dynamic operation is to choose the optimal level of compression based on the reconstructed signal. Such a solution requires feedback from the downstream tasks and thus is not a fit for an IoT setting where the compression and decompression modules are decoupled. To address this, we extend a classical CAE architecture also to learn to predict the downstream tasks' feedback as part of the compressed representation. This prediction eliminates the feedback loop between the IoT devices and the Cloud but still assures an optimal compressed representation that meets desired performance of downstream tasks. To further reduce the computational costs of our model, we apply standard (deep learning) architecture optimization techniques. We believe that our work may be the first to propose a dynamic compression scheme with a tunable downstream task error that easily matches the design requirements for IoT medical applications.

[2]Modern implementations of ECG applications are encoder-based. Our design uses a dedicated encoder for

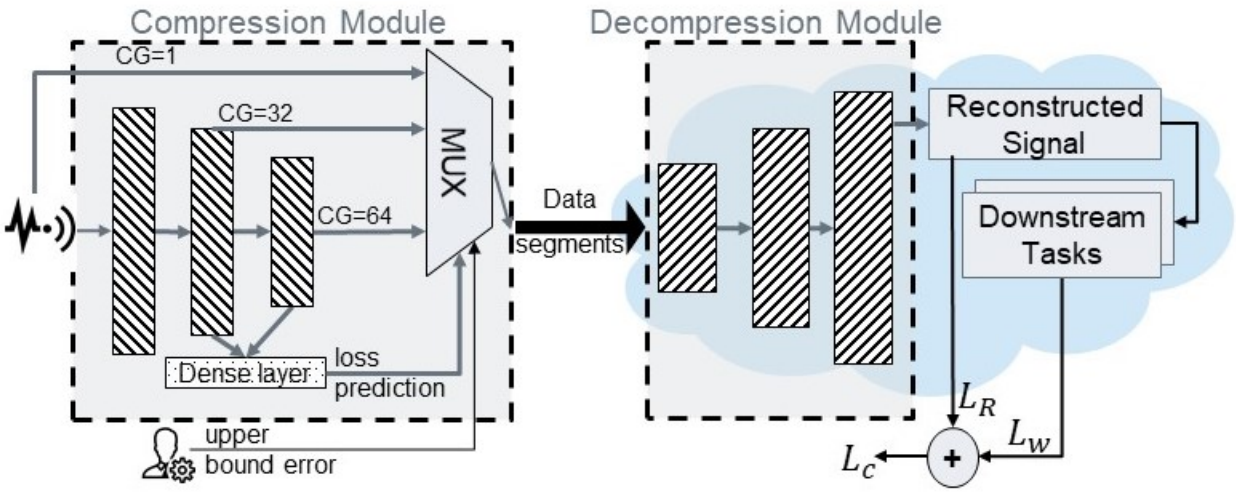


Figure 3: Schematic architecture of Dynamic-Deep. The compression and decompression modules use stacked convolutional layers with downsample and upsample layers, respectively, to achieve multiple compression levels[2]. The encoder uses a dense layer to predict the task error for each level.

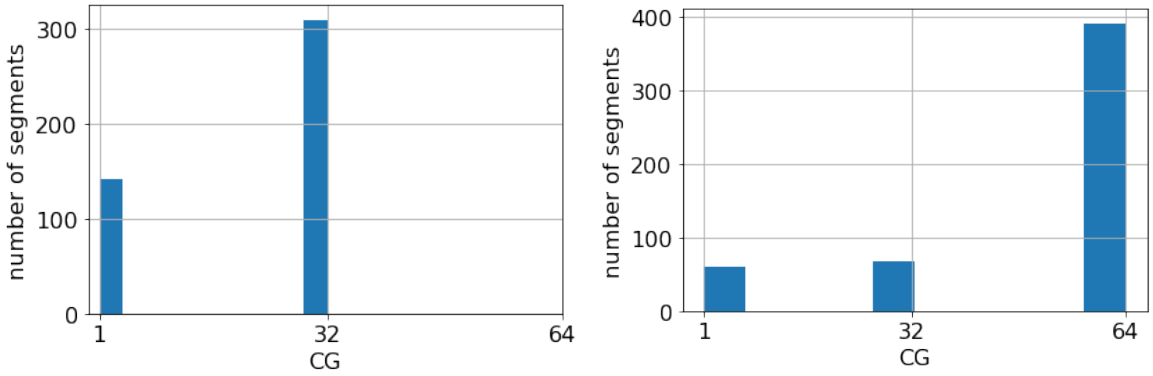
We conducted a comprehensive evaluation of our proposed method using two complex real-life ECG applications, R-R peaks detection (RPNet [23]) and HR arrhythmia classification [19] with widely used ECG dataset. The dataset includes 50 hours ECG recordings with different types of labeled arrhythmia annotated by medical experts. The test set has 602 data segments with 8.9K samples each while 40% contains abnormal events (see full datasets specs in Section 6.1). Our compression method can be configured to wide range of downstream task performances. One of which can increase HR arrhythmia classification F1-score by 3 times, and the average CG by up to 83% (with a maximum CG of 64), compared to a SOTA CAE compression with a fixed CG [26] (Table 1 compare arbitrary working knobs against SOTA CAE). These improvements stem from training the compressor with the downstream task loss feedback (yield better F1-score) and applying a dynamic compression approach (yield better CG). Fig. 4 shows the resulted CG histogram of 2 chosen configured upper bound errors when applying Dynamic-Deep approach. Such approach enable Dynamic-Deep to adapt itself automatically to the pre-configured upper bound and to automatically choose the optimal CG combinations. Dynamic-Deep successfully preserved the desired error bound for all data segments; Whereas SOTA CAE violates the HR arrhythmia classification task's error bound for 10.21% of the data segments. Here, the classification task's error is defined as the categorical cross entropy (CCE) loss, a common loss function for classification

compression, decoupled from the downstream encoder, to allow a lightweight compressor implementation and support cases where the downstream task internals are not available to the admin due to business or IP restrictions.

models. Further, the memory footprint of our solution is 67% lower than that of the CAE. In addition, we deployed our solution on the Google Cloud Platform to study its real-world Cloud costs. Decompressing data traffic equivalent to that of a small-mid-sized hospital, we observe that a reductions in Cloud expenses by up to 97% compared to that of non-compressed data traffic. Section 6 presents the full evaluation as well as the evaluation of additional downstream task RPnet. RPNet detects R-R peaks in ECG and considered being important to precursor to the diagnosis of numerous cardiac disease.

Our method can be easily extended to incorporate additional downstream tasks by adding the corresponding loss functions. Moreover, while we present the results of Dynamic-Deep on medical ECG data, our method is general, and thus can be extended to support other sensor types and potentially other domains by simply adapting the compressor modules [3].

This paper is organized as follows. Section 2 discusses related works on ECG signal compression. Sections 3-5 explain the motivation, the high-level and detailed design of our proposed solution. Next, evaluation experiments are presented in 6. Finally, we conclude in 7.



(a) Low upper bound configuration prevents Dynamic-Deep choosing CG of 64

(b) High upper bound error (Dynamic-Deep #2) configuration

Figure 4: Configuring an upper bound error enable a dynamic compression approach

2 Related Work

Existing compression techniques were adapted into medical IoT environments, typically to fit the low power requirements [22], [9]. Such methods are split into lossless [24] and lossy [9] categories.

[3]Source code available at https://github.com/eladwass/Dynamic-Deep

Lossless achieves low CG on signals, such as ECG [16], while lossy compression is highly efficient (in reducing storage requirements), and thus fit for IoT sensing.

Transform-based compression. A common approach for lossy compression is transform-based, which seeks to preserve the crucial parts of the signal's representation in the transformed domain. Few notable examples are Fourier transform [20], Wavelet transform [4] or the Cosine transform [2], each with its own domain transformation preference. The transformed representation of ECG data is often sparse, and thus preserving the right parts of the representation enables one to get a reconstruction signal of potentially high fidelity [9], [5]. In [22] a proposal for a dynamic scheme that adopts the threshold on the representation size is suggested. The main drawback of the transform-based approach is the use of a predefined domain transformation, which may not lead to the highest CG for the desired reconstruction level.

Neural network (NN) based compression. NNs automate the process of searching for an optimal domain transformation for the compressed representation. Auto-Encoders (AE), a family of NNs, were extensively studied [14], [12] and shown to effectively learn an expressive-yet-efficient representation of ECG segments, and thus provide a higher CG than transform-based compression methods [22]. Recent work [26] uses a convolutional AE with 27 layers to achieve SOTA compression results. In general, AE architectures have a single fixed compression level, and thus are limited in the achievable CGs.

Variable-rate compression. Recent works suggested NN architectures with multiple compression levels that allow balancing between CG and reconstruction quality [7] [3]. However, they require to define in advance a rate control parameter. Additionally, they focus their evaluation on reconstruction quality only, therefore there is no guarantee on downstream tasks performance. Whereas Dynamic-Deep is tuned to a desired downstream tasks' performance and optimizes CG for each data segment automatically rather than manual parameter changes.

Task-aware compression. Recent NN-based data compression architectures consider the joint performance of signal reconstruction and single downstream task, such as Person keypoint detection [18] or 3D point cloud classification tasks [10]. Such methods perform better with respect to downstream tasks performance and compression gain than its standalone single-task counterparts. However, none of them introduced performance analysis of multiple downstream tasks. Dynamic-Deep applies such an approach for the ECG domain and studies how to tune a multiple-level

task-aware compressor to optimize overall CG results.

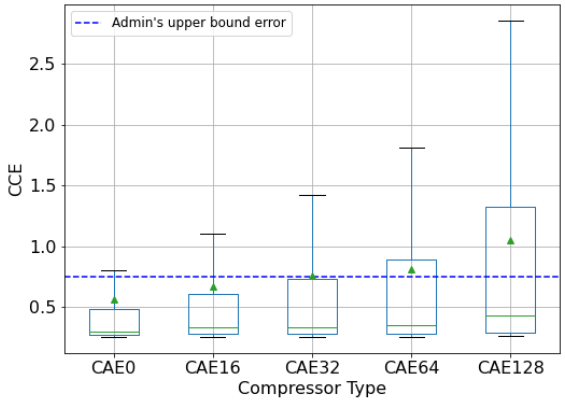
3 Motivation For Balancing CG And Downstream Tasks Performance

Lossy compression methods, such as SOTA CAE [26], are based on an architecture with a single fixed compression level. However, using a fixed compression level may not necessarily satisfy a desired bound on the task's error for every data segment.

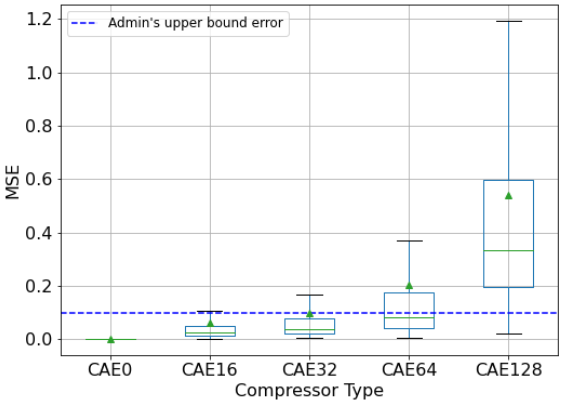
Fig. 5 presents the loss quartiles across 602 ECG test data segments for different tasks and fixed compression levels: 128, 64, 32, 16 and none (See section 6.1 for detailed dataset specs). Denote CAExx as an extended SOTA CAE implementation with a CG of xx, and let CAE0 represents no compression. We note that non zero losses may occur in uncompressed operations due to the inherent error of the downstream task model.

Let an example scenario be when the admin bound the HR arrhythmia classification loss (CCE) to 0.75, none of the fixed compressors can satisfy this bound for all segments, as shown in Fig. 5a. To meet the bound of 0.75 we can apply an approach of 2-compression levels by combining a single compressor and no compression. For instance, with a single compressor CAE32, 75% of data segments meet the upper bound, hence the rest 25% remain non compressed. Such a setting yields an average CG of 24. Additionally, we can leverage higher CG as long as those meet the upper bound error of desired downstream task. For instance, CAE64 meets the upper bound for approximately more than 70% of data segments. Therefore, increasing to 3-compression levels allows more compression possibilities with potential of higher CG. Back to the example meeting the bound of 0.75, approximately 75% of data segments can be compressed with a high CG of 64 or 32 while the rest 25% remain uncompressed, reaching higher CG of 48.31. Increasing the number of compression levels to 4 with maximum CG of 64 and upper bound error of 0.75 improves average CG to 49.2. Finally, with a 5-compression level and maximum CG of 128 almost double the average CG, still, with the upper bound of 0.75 (See further discussions in 6).

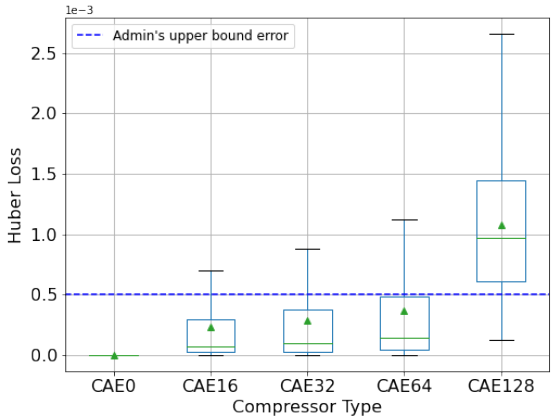
For the reconstruction task, a fixed compression level also does not necessarily satisfy a low configured upper bound error for every data segment. Fig. 5b presents a similar challenge the admin faces with the HR arrhythmia classification task. It is challenging to configure a low upper



(a) HR arrhythmia classification



(b) Reconstruction task



(c) R-R peak extraction (RPNet) task.[4]

Figure 5: The loss's quartiles over test ECG data segments (of size 602 as in [19]) for various tasks.

bound that satisfies a single fixed compression level. But choosing multiple compression levels can solve this challenge as presented for the HR arrhythmia classification task.

As opposed to the HR arrhythmia classification task and reconstruction task, the R-R peak extraction task can benefit less from multiple compression levels. The difference in loss quartiles between compression levels is not significant compared to the former analyzed tasks. Therefore, we expect to see a minor advantage in favor of higher compression levels against lower compression levels (see Fig. 8 for the results). Here, the R-R peak extraction loss is defined as the Huber loss, also known as smooth L_1 loss [23].

Tuning the number of compression levels with related CG (128, 64, 32, 16 or none) comes with a tradeoff of model complexity and performance (see evaluation in section 6.3). Following the evaluation we chose to proceed with 3-compression levels with CAE64, CAE32 and CAE0.

Balancing CG and the downstream tasks' error is challenging, as the compressor needs the task's error feedback for every level of compression to optimize for the highest CG. But, in IoT settings, downstream tasks are decoupled from the compressor and located in the Cloud. We explain how to manage and provide the error feedback in the next section.

4 Dynamic-Deep High Level Design

To receive feedback from the downstream tasks, we propose a solution that extends the classical CAE architecture with multiple compression levels in a task-aware fashion (for discussions regarding the number of levels see section 6.3). Dynamic-Deep consists of the following (as shown in Fig. 3):

1. *Compression Module*: set of encoders that compress every raw data segment into multiple compression level representations. To reduce the need for actual feedback from the downstream tasks, we employ a *dense layer* trained to predict the downstream tasks' weighted error for each compression level. We choose the highest feasible CG based on the pre-configured upper bound error and the feedback prediction.

2. *Decompression Module*: set of decoders that accept multiple representations of different compression levels and reconstruct the data segment.

For optimizing the joint performance of downstream and reconstructed tasks, the training phase requires differentiable downstream tasks (e.g. NN-based). Nonetheless, after the training phase, the reconstructed signal (from the pre-trained compressor) allows executing additional downstream tasks not necessarily differentiable. We found a low correlation between reconstruction error and downstream tasks' error (see Section 5.4). We therefore designed Dynamic-Deep to predict the downstream tasks' error rather than the reconstruction error as a proxy for the error feedback.

5 Dynamic-Deep Implementation Details

5.1 Multiple Compression Levels

The following architectural changes were made to extend the CAE32 to support a CG of 64 and reach 3-compression levels (64,32 or no compression). See appendix B for full detailed network.We chose

[4]CAE0 equals zero since no ground truth exists on CINC dataset

3-compression levels based on experiments comparing the performance (and footprint) of different numbers of compression levels, including CAE16. We obtained highest impact with 3-compression levels (see Section 6.3). Denote conv(x,y) as a convolutional layer with x number of filters, kernel size of y, and Upsample(z) as an upsample layer with size z. To support CG of 64, convolutional layers conv(64,7) and conv(1,3) were added after CAE32 encoder's layer number 12, yielding an output shape of (31,1).

A decoder adapter transforms the compressed representation of 64 to input the CAE32's decoder. The adapter has four layers: conv(16,7), conv(32,3), Upsample(2), conv(32,3), Upsample(2) with an output size of (124,32). The output of the adapter is the input of layer 18 in CAE32. Each compressed representation has its *dense layer* to support the downstream tasks' prediction. The dense layer's output shape is (1,1), and it applies a ReLU activation [1].

5.2 Minimizing Memory Footprint For IoT Support

Deploying compressing modules on lightweight IoT devices requires a resource-constrained implementation. We use two encoders to support 3-compression levels, which results in a 1.06MB memory footprint for the extended SOTA CAE. We use several techniques to reduce the model's memory size:

1. Deep learning compression techniques: The number of learnable parameters of the CAE encoder's 10th layer is reduced by decreasing the number of filters and kernel size. We compensate and preserve compression performance by preserving the receptive field using convolutional layers with stride=2 instead of pooling layers [6]. Note that the encoder's 10th layer is used in all compression levels.

2. Sharing layers: first layers of each encoder (CAE32 and CAE64) have the same learnable parameters, therefore memory and computation are reused and avoid linear memory increase for each compression level [13]. Only the last encoder's layers of each compression level are computed uniquely. Fig. 6 illustrates the final design.

Table 2 summarizes the resulting memory size when Applying the above techniques. It decreases the number of parameters to only 84K parameters which are 67% fewer parameters to the straightforward 3-compression level CAE's encoder.

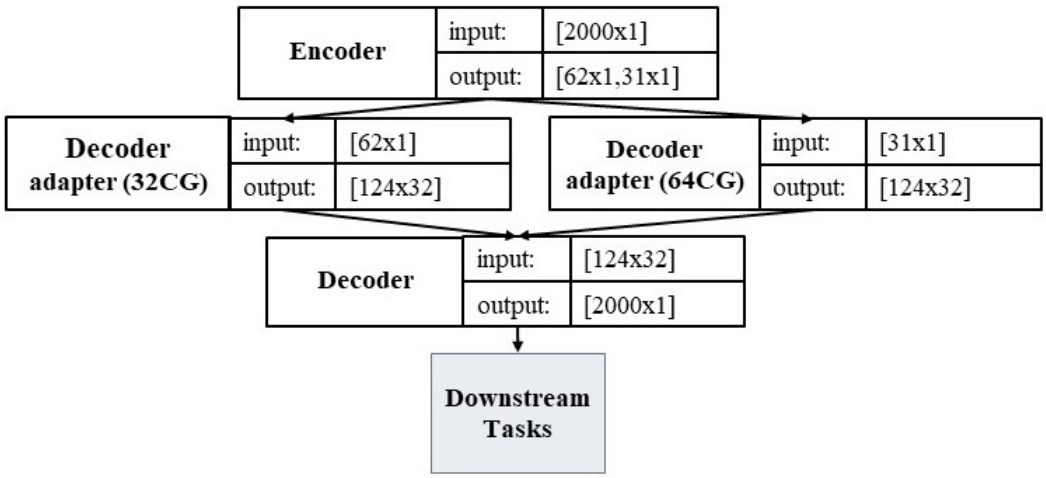


Figure 6: Dynamic-Deep shared layers reduce memory footprint

5.3 Combined Loss Function

Dynamic-Deep has three loss functions accumulated to a combined loss function. The *reconstruction loss* L_R, calculates the distance between each sample in original data segment X and the reconstructed data segment $\hat{X}$ over M samples.

$$L_R = \frac{1}{M} \sum_{i=0}^{M-1} \frac{|X[i] - \hat{X}[i]|}{X[i]} * 100 \tag{1}$$

The *downstream task weighted error* L_w, accumulates the downstream tasks' loss functions. Let t_i denote task i, and L_{t_i} and w_i denote its loss function and the weighted (scaling) factor of the loss function, respectively.

$$L_w = \sum_i w_i * L_{t_i} \tag{2}$$

The *combined loss function* L_c, combines the reconstruction loss L_R with the downstream tasks' weighted error L_w. w_0 scales the L_R to balance between reconstruction performance and downstream

Table 2: Compression Module's Memory minimization

	Number of parameters	memory size [KB]
Original 2-compression level (CAE based)	133K	532
Original 3-compression level (CAE based)	266K	1064
Deep learning compression techniques	168K	678
Sharing layers (**Dynamic-Deep 3-compression level**)	**83K**	**332**

tasks performances.

$$L_c = w_0 * L_R + L_w \tag{3}$$

Finally, Dynamic-Deep learns to predict L_w using the *mean squared error (MSE)*.

5.4 Training

We observe a low correlation between the tested downstream tasks and the reconstruction errors across a wide range of the weighting factors w_i (see Section 6.4). Hence, training a downstream task in isolation on the reconstructed signal may result in limited performance.

We thus train Dynamic-Deep in three phases, where the second phase is repeated for each additional downstream task. First, we train the compressor to optimize the reconstruction task (see eq. (1)). Here, we use the MIT-BIH dataset by applying the preprocessing described in [26]. Second, we fine-tune with cascaded downstream tasks loss, including reconstruction loss (see eq. (3)). The downstream tasks' models' weights are frozen and trained for additional 20 epochs with the CinC dataset[5], by applying the preprocessing described in [19]. Finally, we train the *dense layer* to predict the downstream tasks' weighted error for additional 10 epochs. As before, the compressor is frozen to not penalize previous training phases.

All phases use the Keras API with TensorFlow 2.2 backend, an Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, decay$= 1e-5$, and a batch size of 32.

5.5 Tuning Loss Weighting Factors

Training downstream tasks isolated from the compressor may result in limited performance. Therefore a fine-tuning training phase is applied. The balancing between the downstream tasks and the reconstruction task is achieved using weighted factors w_i in L_c. The weighted factors are hyper-parameters that help reach a working point where the performance degradation is minimal for all downstream tasks and reconstruction tasks.

Increasing w_0, increases the domination of the reconstruction loss and, therefore, improves the reconstruction quality. However, increasing the values of w_i (weights of downstream task t_i) im-

[5]For the essence of evaluation, all training phases use the same CinC dataset. The HR Arrhythmia task use the dataset as-is therefore compared against medical expert. For the RPNet task, we compare the performance of different CG against CG=1 (i.e. no compression)

proves the performance of the downstream task on account of the other tasks. Finally we set $w_0 = 0.04$,$w_1 = 0.015$,$w_2 = 0.4$ for the reconstruction task, HR arrhythmia detection task and R-R peak extraction task respectively. The chosen weights scale the loss components of each task t_i and reconstruction task and prevent the domination of one of them. We believe we can also use the weighting factor to help the admin favor a specific task, but this is outside the scope of this work since it brings an additional challenging dimension of how to interpret downstream task performance with tunable weighting factors.

6 Experimental Results

6.1 Datasets and Downstream Tasks

Datasets: used for training and evaluation:

- *MIT-BIH* [17]: used to evaluate ECG compression as it includes different types of noise patterns and various shapes of arrhythmic QRS complexes [15]. The benchmark contains 48 half-hour ambulatory ECG recordings with 11-bit resolution and sample rate of 360Hz yielding a data set comprising 4800 ECG data segments.

- *CinC* [8]: Captured from the AliveCor ECG monitor and contains about 7000 records with 8960 samples each. These records are annotated by medical expert to the following classes: Atrial Fibrillation(AF), noise, other rhythms or normal. The data was stored at a sample of 300 Hz. The test set has 602 data records where 40% contains abnormal events. Training set has 5403 data records. Note that, each record contains multiple sliding data segment of 2000 samples which is the input size of Dynamic-Deep.

Downstream tasks: two kind of tasks (NN based) architectures were chosen:

- HR arrhythmias classification: implemented using convolutional NN (CNN) [19] as a classification task. The ground truth of this task uses labels from the CinC dataset. Those labels are annotated by medical experts.

- R-R peak extraction(RPnet): implemented using NN [23] as a regression task locating the position in time of the peak. The ground truth of this task (i.e. R-R peaks on raw ECG) is

generated by running the RPnet NN on the raw ECG signals offered by the CinC dataset. Then different CG were evaluated relatively to the ground truth.

Both are commonly used tasks in real-world ECG applications.

6.2 Task Awareness Evaluation

We focus our evaluation on the Dynamic-Deep downstream tasks' predictions performed by the *dense layer* in the compression module. We compare our predictions against two theoretical models (see why in 6.3), in which the downstream task feedback is available for the compressor:

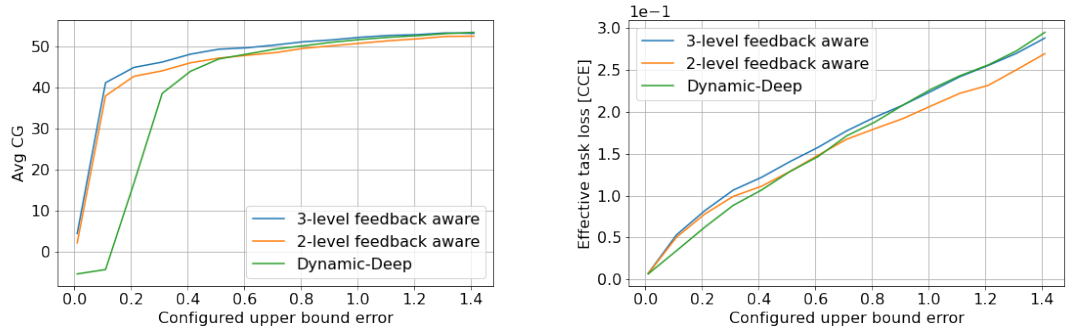
1. *2-level feedback-aware*: the method has 2-compression levels of 64 or no compression.
2. *3-level feedback-aware*: the method has 3-compression levels of 64, 32 or no compression.

Each model executes the downstream tasks for every data segment and measures the error at each compression level. Then, they choose the highest compression that meets the configured upper bound error. If none meets the upper bound, the no compression level is chosen. Note that such a method is not applicable in typical IoT settings since the feedback is not readily available at the edges. We evaluated Dynamic-Deep vs. the theoretical models above considering the following setups:

1. *Single downstream task awareness*: of HR arrhythmia classification or R-R peak extraction. Fig. 7 shows that Dynamic-Deep follows the trends of the theoretical method successfully. Increasing the upper bound error increases the CG and the effective task loss and vice versa. For every configured upper bound Dynamic-Deep results with a lower effective task loss than the configured upper bound. There is an improvement in CG for both tasks when increasing the number of compression levels from 2 to 3.

   For the RPNet task, moving from 2 to 3 compression levels yields a diminishing return. The diminishing return is a result of low difference in loss quartiles between compression levels as explained in section 3. Nonetheless, Dynamic-Deep successfully follows the trends and chooses the optimal decision (see fig. 8).

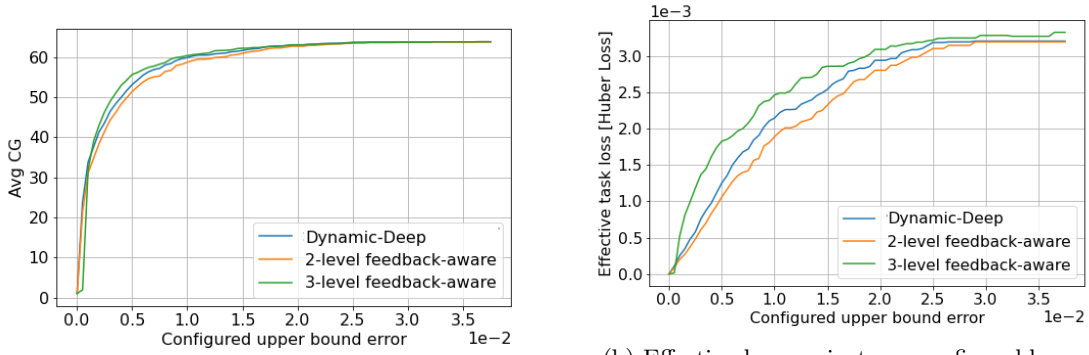
2. *Multiple downstream tasks' awareness*: of both R-R peak extraction and HR arrhythmia classification. Supporting multiple downstream tasks introduces a tradeoff on which downstream



(a) Average CG against pre-configurable bound

(b) Effective loss against pre-configurable bound

Figure 7: Benchmark on HR arrhythmia classification task shows Dynamic-Deep successfully balance between desired performance and CG



(a) Average CG against pre-configurable upper bound

(b) Effective loss against pre-configurable upper bound

Figure 8: Benchmark on RPNet (R-R extraction) shows Dynamic-Deep successfully balance between desired performance and CG

task to optimize. Dynamic-Deep uses a downstream task weighted error (see eq. 2), which can be viewed as a single task awareness. This setup achieve similar results as single downstream task awareness

When designing Dynamic-Deep, two additional parameters are taken into account: the number of compression levels and the CG of each compression level. The following section presents different experiments that helped design the final architecture of Dynamic-Deep.

6.3 Tuning Compression Level And Compression Gain

To set Dynamic-Deep's parameters (number of compression levels and their compression gain), a grid search process is applied to optimize for the highest CG and the highest downstream tasks' performance. To save training time, theoretical feedback-aware models is used to approximate

Dynamic-Deep performance. The best performed parameters are used to train Dynamic-Deep. We compared the following theoretical feedback-aware models:

1. 2-level feedback-aware: the method has 2-compression levels of 64 or no compression
2. 3-level feedback-aware: the method has 3-compression levels of 64, 32 or no compression
3. 4-level feedback-aware: the method has 4-compression levels of 64, 32, 16 or no compression
4. 5-level feedback-aware: the method has 5-compression levels of 128, 64, 32, 16 or no compression

We applied the following phases to choose between the above models:

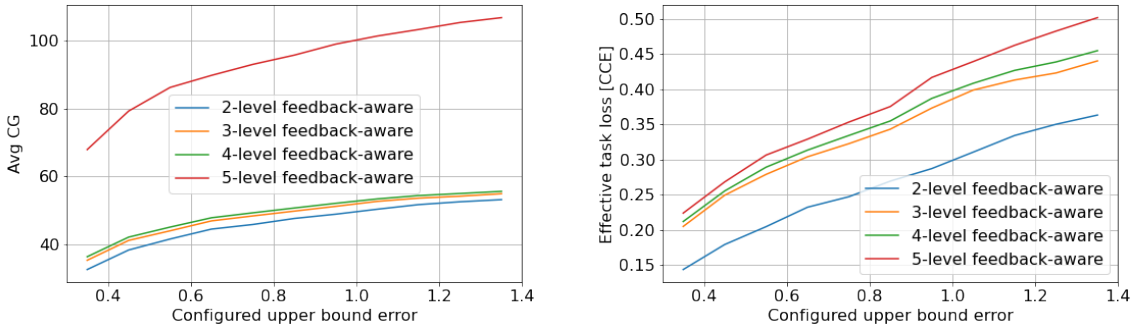
1. Set an upper bound CG: that still allow functional operation of the end to end system. The desired upper bound CG allows reconstruct the signal, allow non-noisy outputs from the downstream tasks and allow learning to predict downstream tasks error.
2. Choose number of compression levels: after setting the upper bound CG, we add compression levels till we see saturation in CG performance.

Fig. 9 presents the performance of each model on the HR arrhythmia classification task. 5-level feedback-aware with an upper bound CG of 128 succeeds in doubling the CG compare to other setups. However, the CG of 128 expressed by a compressed representation of 64 bytes, failed to learn the predictions of the downstream task feedback. The training attempts reached an overfitting and did not generalize to unknown signals. We believe a compressed representation of 64 bytes is too small to succeed in the predicting task. Thus we did not choose models with an upper bound CG of 128 and tuned the upper bound CG to 64.

Increasing the number of compression levels from 2 to 3 has higher performance gain as opposed to changing from 3 to 4. One can think of simply choosing the configuration with the highest compression levels, but additional compression levels come with the price of model complexity. Any additional compression level demand in average additional 2K parameters, depend on the CG of the added level. Therefore the choice of 3 compression levels is just before reaching saturation in CG and avoids the complexity of higher CG implementation. For the R-R peak extraction task, Fig. 8

presents that the increase from 2 to 3 compression levels is minor and almost neglected. Therefore a 2 compression level for this task reaches a good balance.

To sum up, Dynamic-Deep is designed with a 3-compression level to align with the task requiring the highest number of compression levels.



(a) Average CG against pre-configurable bound

(b) Effective loss against pre-configurable bound

Figure 9: Theoretical feedback-aware comparison with different compression level for the HR arrhythmia classification task

6.4 Low Correlation Between Downstream Tasks and Reconstruction Error

We measured a weak correlation (equal to 0.23) between reconstruction loss and the downstream tasks' loss using Spearman metric [25]. Moreover, when changing the weighting factor w_i of L_c in favor of optimizing a specific downstream task, the correlation is reduced by almost half. We suspected the existence of low correlation following two symptoms:

1. Lower performance when feedback-awareness using reconstruction loss: the initial design used the reconstruction error as feedback to choose a compression level. However, fig. 10 presents that optimizing based on downstream task loss (e.g. HR arrhythmia classification loss) reduced the effective loss up to 70% with the same CG.

2. Fine-tuning pre-trained compressor: We expected that optimizing only the reconstruction task would meet the similar performance as fine-tuning a pre-trained compressor. However, fine-tuning improved the (tested) downstream tasks' performance.

Following the above experiments, we designed Dynamic-Deep to predict the downstream task error instead of using the reconstruction error.

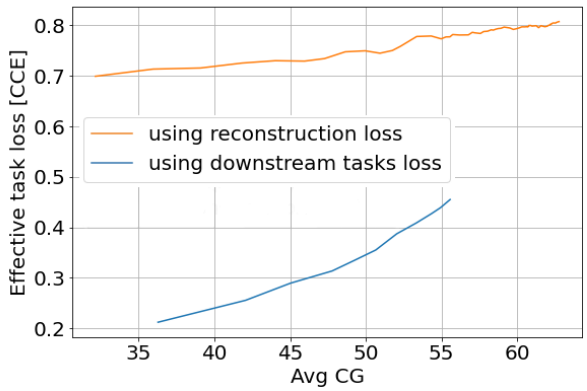


Figure 10: Compare different feedback-awareness strategies using HR arrhythmia classification loss

6.5 Cloud Cost Reduction Analysis Using Dynamic-Deep

Compression methods are used in IoT settings to reduce storage costs and networking bandwidth. For Cloud costs evaluation, we consider storage and computation (data decompression for consumption by downstream tasks) costs since Cloud inbound traffic is usually free. We compare the following operational models:

1. *Dynamic-Deep*: IoT device sends a compressed representation, which is stored in the Cloud, and Cloud side decompression phase is applied before downstream tasks' execution.

2. *Dynamic-Deep with uncompressed*: IoT device sends both compressed and uncompressed representations. The compressed representation is stored while downstream tasks operate on the uncompressed representation.

We assume that a domain expert reviews some portion $x\%$ of historical sensor data, and accounts for the corresponding overhead, the cost of fetching data from storage and decompressing it, in both models.

We ran these two models on Google Cloud Platform [11] using ECG data segment traffic equivalent to a small-mid hospital with 200 beds. We configured the upper bound error of HR arrhythmia classification to be 0.75 and received an average CG of 48.31. We measured the computation expenses of our setup on an N1-Custom instance with 1 CPU, 2GB RAM, Intel Xeon 2.2GHz. Fig. 11 presents the measured results. Lossless compression reduces expenses by 63% regardless of the specific architecture due to its low computation usage. *Dynamic-Deep with uncompressed* architecture

saves up to 97% cost expenses compared to no compression solution and is more efficient than lossless even with 100% data fetching.

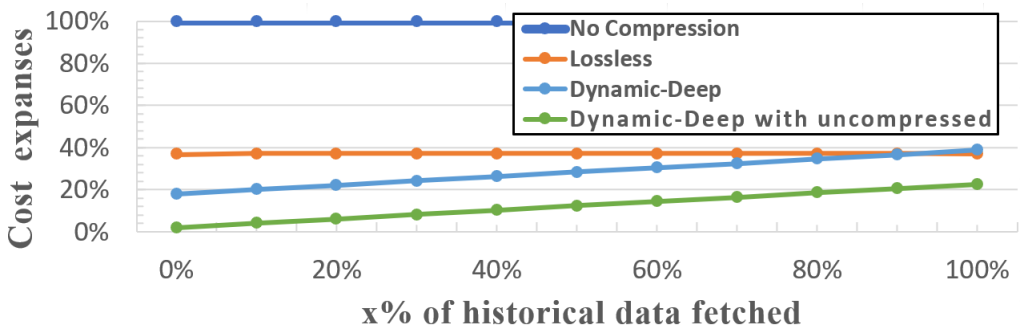


Figure 11: Yearly cost expanses compression comparison

7 Conclusion

We presented a variable-rate compression system suitable for IoT-Cloud applications. The method leverages downstream task error prediction at the edge to overcome challenges within Cloud architectures and to allow the system admin to configure a desired performance level. We successfully showed CG improvements on two types of downstream tasks against a SOTA CAE. Additionally we showed the method allows the practitioner to balance between desired performance and compression gain. Future work will extend the implementations to other domains.

References

[1] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[2] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[3] M. Akbari, J. Liang, J. Han, and C. Tu, "Learned variable-rate image compression with residual divisive normalization," in *2020 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2020, pp. 1–6.

[4] J. Chen, S. Itoh, and T. Hashimoto, "Ecg data compression by using wavelet transform," *IEICE TRANSACTIONS on Information and Systems*, vol. 76, no. 12, pp. 1454–1461, 1993.

[5] A. F. Cheng, S. E. Hawkins III, L. Nguyen, C. A. Monaco, and G. G. Seagrave, "Data compression using chebyshev transform," 2007.

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[7] Y. Choi, M. El-Khamy, and J. Lee, "Variable rate deep image compression with a conditional autoencoder," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3146–3154.

[8] G. D. Clifford, I. Silva, B. Moody, Q. Li, D. Kella, A. Shahin, T. Kooistra, D. Perry, and R. G. Mark, "The physionet/computing in cardiology challenge 2015: reducing false arrhythmia alarms in the icu," in *2015 Computing in Cardiology Conference (CinC)*. IEEE, 2015, pp. 273–276.

[9] H. Djelouat, A. Amira, and F. Bensaali, "Compressive sensing-based iot applications: A review," *Journal of Sensor and Actuator Networks*, vol. 7, no. 4, p. 45, 2018.

[10] O. Dovrat, I. Lang, and S. Avidan, "Learning to sample," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2760–2769.

[11] Google. Machine types,compute engine documentation, google cloud. [Online]. Available: cloud.google.com/compute/docs/machine-types

[12] R. Goroshin and Y. LeCun, "Saturating auto-encoders," *arXiv preprint arXiv:1301.3577*, 2013.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] S. Hong, Y. Zhou, J. Shang, C. Xiao, and J. Sun, "Opportunities and challenges of deep learning methods for electrocardiogram data: A systematic review," *Computers in Biology and Medicine*, p. 103801, 2020.

[16] A. Koski, "Lossless ecg encoding," *Computer Methods and Programs in Biomedicine*, vol. 52, no. 1, pp. 23–33, 1997.

[17] R. Mark and G. Moody, "Mit-bih arrhythmia database directory," *Cambridge: Massachusetts Institute of Technology*, 1988.

[18] R. Pinkham, T. Schmidt, and A. Berkovich, "Algorithm-aware neural network based image compression for high-speed imaging," in *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. IEEE, 2020, pp. 196–199.

[19] P. Rajpurkar, A. Y. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng, "Cardiologist-level arrhythmia detection with convolutional neural networks," *arXiv preprint arXiv:1707.01836*, 2017.

[20] B. S. Reddy and I. Murthy, "Ecg data compression using fourier descriptors," *IEEE Transactions on Biomedical Engineering*, no. 4, pp. 428–434, 1986.

[21] C. B. A. Sampath, "Ecg signal analysis for myocardial infarction detection," 2013.

[22] A. Ukil, S. Bandyopadhyay, and A. Pal, "Iot data compression: Sensor-agnostic approach," in *2015 Data Compression Conference*. IEEE, 2015, pp. 303–312.

[23] S. Vijayarangan, R. Vignesh, B. Murugesan, S. Preejith, J. Joseph, and M. Sivaprakasam, "Rpnet: A deep learning approach for robust r peak detection in noisy ecg," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2020, pp. 345–348.

[24] T. A. Welch, "A technique for high-performance data compression," *Computer*, no. 6, pp. 8–19, 1984.

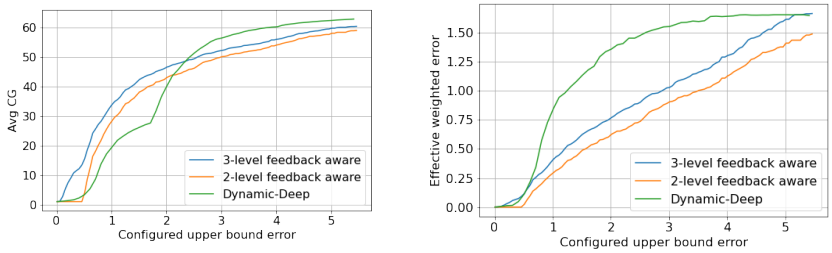
[25] C. Wissler, "The spearman correlation formula," *Science*, vol. 22, no. 558, pp. 309–311, 1905.

[26] O. Yildirim, R. San Tan, and U. R. Acharya, "An efficient compression of ecg signals using deep convolutional autoencoders," *Cognitive Systems Research*, vol. 52, pp. 198–211, 2018.

Appendices

A Evaluate Multiple Downstream tasks

Fig 12 presents the awareness Dynamic-Deep has for both R-R peak extraction and HR arrhythmia classification. Supporting multiple downstream tasks introduces a tradeoff on which downstream task to optimize. Dynamic-Deep uses a downstream task weighted error, which can be viewed as a single task awareness. Fig. 12 demonstrates that multi-task awareness has similar results to a single task awareness.



(a) Average CG against pre-configurable bound

(b) Effective error against pre-configurable bound

Figure 12: Benchmark on multiple downstream tasks shows Dynamic-Deep successfully balance between desired performance and CG

B Dynamic-Deep Low Level implementation details

The following figures detail the low level implementation details and parameters used in each layer of Dyamic-Deep. Denote conv1d_x_y_activationRRRR as a one dimensional convolution with x filters, y size of kernel with activation function. The four RRRR indicate a unique number for each new instance of layer.

1. High Level design: Fig. 13 details the main component of compressor (indicated as Shared_Encoder) and the Decompressor (indicated as decoder)

2. Compressor: Fig. 14 presents the layers used in the compressor and is called as Shared_Encoder

3. Decompressor: Figures 15 and 16 details all layers used to impement the decompressor

4. *dense layer*: Fig. 17 detail the layers used to implement the prediction of the downstream tasks' error

```
Model: "U-NET_32_64_iot"
_________________________________________________________________________________________________________________________________
Layer (type)                                 Output Shape                  Param #           Connected to
=================================================================================================================================
encoder_input_combined (InputLayer)          [(None, 2000, 1)]             0
_________________________________________________________________________________________________________________________________
Shared_Encoder (Encoder_32_64_iot)           ((None, 63, 1), (None, 32, 1))  83898           encoder_input_combined[0][0]
_________________________________________________________________________________________________________________________________
zero_padding1d_1 (ZeroPadding1D)             (None, 95, 1)                 0                 Shared_Encoder[0][0]
_________________________________________________________________________________________________________________________________
zero_padding1d_2 (ZeroPadding1D)             (None, 95, 1)                 0                 Shared_Encoder[0][1]
_________________________________________________________________________________________________________________________________
decoder (Functional)                         (None, 2000, 1)               4094138           zero_padding1d_1[0][0]
                                                                                             zero_padding1d_2[0][0]
=================================================================================================================================
Total params: 4,178,036
Trainable params: 4,177,940
Non-trainable params: 96
_________________________________________________________________________________________________________________________________
```

Figure 13: High Level Modules

```
Model: "Shared_Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_8_3_swish16414 (Conv1D)  multiple                  32
_________________________________________________________________
conv1d_16_3_swish15609 (Conv1D)  multiple                  400
_________________________________________________________________
conv1d_8_1_swish14806 (Conv1D)  multiple                  136
_________________________________________________________________
conv1d_32_5_swish15432 (Conv1D)  multiple                  1312
_________________________________________________________________
batch_normalization (BatchNormalization)  multiple                  128
_________________________________________________________________
conv1d_64_5_swish14512 (Conv1D)  multiple                  10304
_________________________________________________________________
conv1d_32_1_swish19285 (Conv1D)  multiple                  2080
_________________________________________________________________
conv1d_16_3_swish15233 (Conv1D)  multiple                  1552
_________________________________________________________________
conv1d_32_3_swish14297 (Conv1D)  multiple                  1568
_________________________________________________________________
conv1d_16_1_swish12612 (Conv1D)  multiple                  528
_________________________________________________________________
batch_normalization_1 (BatchNormalization)  multiple                  64
_________________________________________________________________
conv1d_32_11_swish16097 (Conv1D)  multiple                  5664
_________________________________________________________________
conv1d_64_7_swish11491 (Conv1D)  multiple                  14400
_________________________________________________________________
conv1d_64_1_swish13892 (Conv1D)  multiple                  4160
_________________________________________________________________
conv1d_32_7_swish18194 (Conv1D)  multiple                  14368
_________________________________________________________________
conv1d_64_5_swish10310 (Conv1D)  multiple                  10304
_________________________________________________________________
conv1d_32_1_swish19546 (Conv1D)  multiple                  2080
_________________________________________________________________
conv1d_1_7_swish10503 (Conv1D)  multiple                  225
_________________________________________________________________
conv1d_64_7_swish12228 (Conv1D)  multiple                  14400
_________________________________________________________________
conv1d_1_3_swish12466 (Conv1D)  multiple                  193
=================================================================
Total params: 83,898
Trainable params: 83,802
Non-trainable params: 96
_________________________________________________________________
```

Figure 14: Shared encoder transforming input signal into 2-compression level

```
Model: "decoder"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================================================
encoder_input_combined (InputLayer) [(None, 95, 1)]    0
_________________________________________________________________________________________________
lambda (Lambda)              (None, 63, 1)             0         encoder_input_combined[0][0]
_________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 32, 1)             0         encoder_input_combined[0][0]
_________________________________________________________________________________________________
conv1d_8_3_swish12945 (Conv1D) (None, 63, 8)           32        lambda[0][0]
_________________________________________________________________________________________________
conv1d_16_7_swish18616 (Conv1D) (None, 32, 16)         128       lambda_1[0][0]
_________________________________________________________________________________________________
conv1d_16_3_swish17483 (Conv1D) (None, 63, 16)         400       conv1d_8_3_swish12945[0][0]
_________________________________________________________________________________________________
conv1d_32_3_swish15909 (Conv1D) (None, 32, 32)         1568      conv1d_16_7_swish18616[0][0]
_________________________________________________________________________________________________
up_sampling1d (UpSampling1D) (None, 126, 16)           0         conv1d_16_3_swish17483[0][0]
_________________________________________________________________________________________________
up_sampling1d_1 (UpSampling1D) (None, 64, 32)          0         conv1d_32_3_swish15909[0][0]
_________________________________________________________________________________________________
conv1d_16_3_swish10339 (Conv1D) (None, 126, 16)        784       up_sampling1d[0][0]
_________________________________________________________________________________________________
conv1d_32_3_swish14151 (Conv1D) (None, 62, 32)         3104      up_sampling1d_1[0][0]
_________________________________________________________________________________________________
conv1d_32_3_swish10982 (Conv1D) (None, 124, 32)        1568      conv1d_16_3_swish10339[0][0]
_________________________________________________________________________________________________
up_sampling1d_2 (UpSampling1D) (None, 124, 32)         0         conv1d_32_3_swish14151[0][0]
_________________________________________________________________________________________________
Shared_Decoder (Decoder_shared_iot) (None, 2000, 1)    4086554   conv1d_32_3_swish10982[0][0]
                                                                 up_sampling1d_2[0][0]
_________________________________________________________________________________________________
tf.__operators__.add (TFOpLambda) (None, 2000, 1)      0         Shared_Decoder[0][0]
                                                                 Shared_Decoder[1][0]
=================================================================================================
Total params: 4,094,138
Trainable params: 4,094,138
Non-trainable params: 0
_________________________________________________________________________________________________
```

Figure 15: Adapting different compression level to a single shared decoder (decoder_32_64_shared_iot), see Fig. 16

```
Model: "Shared_Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
up_sampling1d_3 (UpSampling1D)  multiple                  0
_________________________________________________________________
zero_padding1d (ZeroPadding1D)  multiple                  0
_________________________________________________________________
conv1d_64_7_swish16024 (Conv1D)  multiple                 14400
_________________________________________________________________
conv1d_128_7_swish10165 (Conv1D)  multiple                57472
_________________________________________________________________
up_sampling1d_4 (UpSampling1D)  multiple                  0
_________________________________________________________________
conv1d_16_3_swish15801 (Conv1D)  multiple                 6160
_________________________________________________________________
conv1d_32_3_swish13585 (Conv1D)  multiple                 2592
_________________________________________________________________
up_sampling1d_5 (UpSampling1D)  multiple                  0
_________________________________________________________________
conv1d_32_3_swish17623 (Conv1D)  multiple                 3104
_________________________________________________________________
conv1d_8_3_swish13532 (Conv1D)  multiple                  776
_________________________________________________________________
conv1d_2_3_swish10743 (Conv1D)  multiple                  50
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
outputs_decoder (Dense)      multiple                  4002000
=================================================================
Total params: 4,086,554
Trainable params: 4,086,554
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: Shared decoder reconstruct the signal back

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_218 (Conv1D)          (None, 252, 4)            16
_________________________________________________________________
conv1d_219 (Conv1D)          (None, 252, 8)            104
_________________________________________________________________
conv1d_220 (Conv1D)          (None, 126, 8)            200
_________________________________________________________________
conv1d_221 (Conv1D)          (None, 126, 4)            100
_________________________________________________________________
conv1d_222 (Conv1D)          (None, 126, 8)            104
_________________________________________________________________
conv1d_223 (Conv1D)          (None, 63, 8)             200
_________________________________________________________________
conv1d_224 (Conv1D)          (None, 63, 1)             25
_________________________________________________________________
dropout_19 (Dropout)         (None, 63)                0
_________________________________________________________________
dense_5 (Dense)              (None, 16)                1024
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,790
Trainable params: 1,790
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: *dense layers* implementation that learns to predict downstream task loss

תקציר

אחת מן האפליקציות הנפוצות ביותר בארכיטקטורות Internet of Things (IoT), הינה ניטור חולים בעזרת אותות רפואיים כדוגמת אלקטרוקרדיוגרם (אק"ג). תהליכי ניטור אלו מייצרים כמויות אדירות של מידע ולכן לרוב משלבים פתרונות דחיסת נתונים עוד בשלב יצירת האות וזאת לטובת צמצום עלויות ענן. פתרון מבוסס lossy compression אמנם מאפשר ביצועי דחיסה גבוהים אך נוטה לפגוע באיכות האות בשל איבוד מידע, וכתוצאה מכך נפגעות ביצועי האפליקציות בענן. עבודות קודמות בנושאי אותות אק"ג מתמקדות, מצד אחד, באופטימיזציה של שיחזור האות לאחר תהליכי דחיסה ומצד שני, באופטימיזציה של ביצועי אפליקציות אק"ג בענן. לעומתם, בעבודה זו ביקשנו לאזן בין השניים ולהציע פתרון מבוסס lossy compression, המאפשר להגדיר גבול עליון לשגיאות בביצועי האפליקציות אק"ג בענן תוך אופטימיזציה לביצועי הדחיסה, וזאת כדי להביא לצמצום בעלויות הענן.

אנחנו מציעים את Dynamic-Deep, פתרון דחיסה שלוקח בחשבון את ביצועי האפליקציות ומותאם לארכיטקטורות ענן. לאחר תהליך האימון, צד ה-IoT מתאים את ביצועי הדחיסה בצורה דינמית עבור כל הודעה ושולח הודעות דחוסות לענן תוך התחשבות בהגדרות ביצועי אפליקציות הענן וללא כל תקשורת עימם.

הפתרון נוסה על מאגרי מידע מוכרים ועל משימות אק"ג פופולריות כמו היכולת לזהות הפרעות בקצב לב. ביחס לפתרון הדחיסה החדיש ביותר המבוסס רשתות נוירונים, הראינו כי הצלחנו לשפר את F1-Score בלפחות פי 3 ולשפר את ביצועי הדחיסה עד רמת של 83%. ניתוח תוצאות הפתרון על תשתית שירותי הענן של גוגל הראו חיסכון של 97% ביחס לאי שימוש בפתרון דחיסה בכלל.

למיטב ידיעתנו, Dynamic-Deep הינה ההצעה הראשונה שמשלבת ומאזנת בין הצורך לביצועים גבוהים של האפליקציות על שירותי הענן והרצון לשמור על ביצועי דחיסה אופטימליים בארכיטקטורות ניטור אק"ג ב- IoT.

עבודה זו בוצעה בהדרכתם של ' **דר' אלי ברוש ופרופ' ענת ברמלר-בר** מבי"ס אפי ארזי למדעי המחשב, אוניברסיטת רייכמן

אוניברסיטת רייכמן
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

Dynamic-Deep:
שליטה בהוצאות ענן וביצועי אפליקציות אק"ג

מאת

אלעד וסרשטיין

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, אוניברסיטת רייכמן

מרץ, 2022