

Progressive Neural Compression for Adaptive Image Offloading under Timing Constraints

Ruiqi Wang, Hanyang Liu, Jiaming Qiu, Moran Xu, Roch Guérin, Chenyang Lu

Department of Computer Science and Engineering

Washington University in St. Louis

St. Louis, Missouri, USA

Email: {ruiqi.w, hanyang.liu, qiujiaming, moranxu, guerin, lu}@wustl.edu

Abstract—IoT devices are increasingly the source of data for machine learning (ML) applications running on edge servers. Data transmissions from devices to servers are often over local wireless networks whose bandwidth is not just limited but, more importantly, variable. Furthermore, in cyber-physical systems interacting with the physical environment, image offloading is also commonly subject to timing constraints. It is, therefore, important to develop an adaptive approach that maximizes the inference performance of ML applications under timing constraints and the resource constraints of IoT devices. In this paper, we use image classification as our target application and propose *progressive neural compression (PNC)* as an efficient solution to this problem. Although neural compression has been used to compress images for different ML applications, existing solutions often produce fixed-size outputs that are unsuitable for timing-constrained offloading over variable bandwidth. To address this limitation, we train a multi-objective rateless autoencoder that optimizes for multiple compression rates via stochastic taildrop to create a compression solution that produces features ordered according to their importance to inference performance. Features are then transmitted in that order based on available bandwidth, with classification ultimately performed using the (sub)set of features received by the deadline. We demonstrate the benefits of PNC over state-of-the-art neural compression approaches and traditional compression methods on a testbed comprising an IoT device and an edge server connected over a wireless network with varying bandwidth.

Index Terms—neural compression, edge offloading, image classification, real-time transmission

I. INTRODUCTION

There is no denying the emergence of deep learning (DL) models broadens the scope of potential solutions for a variety of inference tasks in applications such as computer vision (CV) [1]–[3] and natural language processing (NLP) [4], [5]. However, while much progress has been made in reducing the computational footprint of those models [6], [7], they remain beyond the capabilities of most embedded systems, particularly low-end devices such as sensors. As a result, bringing the capabilities of DL models to bear on tasks that arise in many Internet-of-Things (IoT) deployments has typically been realized through edge computing solutions. This setting is the focus of this paper that targets an image classification task performed in an edge server based on data acquired through a set of distributed sensors.

Although an advantage of edge computing is the relative proximity of compute resources with data sources (sensors),

there is still a need for delivering the data from where they are captured to the edge servers. When dealing with images, as we do in this paper, the sheer size of each data unit commonly calls for some form of compression due to the limited bandwidth of low-power networks such as Zigbee and LoRa.

Traditional image compression schemes are, however, designed with perceptual quality [8] as their target, and, as we shall see, may perform poorly when the metric of interest is instead classification accuracy as in this paper. This has led to much recent interest in compression schemes that are instead optimized for a given inference task. In particular, a number of solutions based on deep neural networks (DNNs) have been proposed [9]–[13] that explicitly incorporate inference accuracy in their design.

These solutions, while effective, are limited by their inability to meet the timing constraint for image offloading under variable bandwidth. As real-time systems incorporate ML, inference tasks often need to complete by a certain deadline, *e.g.*, the arrival of the next image or to allow actuation to be performed in time. For example, consider a warehouse using image classification to automatically distribute packages to different lines. It is important for the image classification task to be completed before the arrival of the next package. The deadline for image classification in turn bounds the time available for image offloading. To meet offloading deadlines, it is essential for the system to *adapt* to the bandwidth variations common to wireless networks [14], [15]. Under the varying bandwidth, the amount of data transmitted by the deadline varies. The implication for any compression and inference scheme is, therefore, that it must be able to classify an image using only the amount of data actually received by the deadline. More generally, the goal is to maximize classification accuracy given any amount of data received by the deadline. Unfortunately, most prior approaches to neural compression generate a fixed amount of data, which must all be received for inference to be performed with sufficient accuracy.

Towards addressing this limitation, we propose a framework called *Progressive Neural Compression (PNC)*. Our approach tackles the problem of progressive offloading by training a rateless compressive autoencoder (AE) that serves multiple objectives. Each objective focuses on optimizing the inference performance based on a different amount of data (partially)

received, which corresponds to a different compression rate. In our methodology, we leverage the “taildrop” technique [16] in recent advances of neural dimensionality reduction to train the multi-objective AE that facilitates progressive image transmissions for classification. Specifically, classification can be performed at any time based only on the data received up to that point, with the receipt of more data translating into progressively more accurate classification decisions. This “progressive” property allows the system to adapt to variations in link bandwidth by continuously offloading data until reaching its offloading deadline. This property is realized by training the AE to ensure that the latent features at its bottleneck layer are sorted in the order of their *importance* for the classification task, *e.g.*, earlier features have higher contributions towards accurate classification. Features are then transmitted in the order of their importance till the deadline. Classification can be performed on the edge server based on the subset of received features, with the classification accuracy improving as more features are received. The progressive nature of PNC allows it to handle different deadlines and varying bandwidth effectively.

The main contributions of this paper are as follows:

- We propose PNC¹, an efficient progressive neural compression framework tailored for IoT devices that need to offload images to edge servers for classification within a deadline despite varying network bandwidth.
- We designed and implemented an end-to-end image classification system integrating time-bounded progressive image offloading and edge-based classification on an edge computing testbed.
- We demonstrated empirically that PNC outperformed both traditional and neural compression solutions in classification accuracy under timing constraints and external wireless interference while realizing a much smaller computational footprint than the latter.

II. BACKGROUND AND MOTIVATION

IoT devices are usually connected to edge servers by low-power wireless sensor networks (*e.g.*, IEEE 802.15.4 or LoRa) with limited bandwidth. Before offloading an image from the device to the server, the image needs to be compressed while still allowing the edge server to classify it accurately. Moreover, these networks typically experience significant fluctuations in available bandwidth due to environmental factors and interference in the wireless channel [17]–[19]. Furthermore, image classification in such settings is often subject to timing constraints. For example, a quality control camera needs to catch a defective item before the arrival of the next item, or a face recognition camera needs to react quickly to prevent unauthorized entry.

The combination of variable and limited bandwidth and time constraints has important implications for image offloading. It impacts the amount of data that can be transmitted

¹The source code and testbed setup instructions for PNC are available at <https://github.com/rickywrq/Progressive-Neural-Compression>.

before classification has to be made. As a result, any edge-based image classification system must be able to adapt to fluctuations in the available bandwidth. One option is to rely on a feedback mechanism that detects bandwidth variations and responds by adjusting compression ratios accordingly. The main disadvantage is the inherent latency associated with having to detect variations before being able to react to them. As a result, we focus on “open-loop” solutions that optimize classification performance given the amount of data transmissions the network allows by the deadline.

We identify the following requirements for an image offloading framework for such image classification:

- **Classification accuracy:** In contrast to traditional image compression approaches optimized for image reconstruction, an offloading framework designed for image classification should target classification accuracy as the primary objective of interest to the application.
- **Encoding efficiency:** Given the stringent resource constraints of IoT devices, the encoding process of the compression approach should incur minimal computation overhead on the devices.
- **Adaptation to different deadlines and varying bandwidth:** Classification accuracy should improve the more data are received by the deadline while achieving graceful performance degradation in the face of decreasing bandwidth.

III. RELATED WORK

Image compression aims at minimizing the size of image representation by removing redundant and irrelevant information [8]. In this section, we review related works on image compression, with a focus on the requirements introduced in Section II.

A. Traditional Image Compression

Traditional compression techniques aim to minimize the amount of data needed to reconstruct an image. For example, JPEG [20], [21], one of the most widely used image compression standards, works by performing discrete cosine transform (DCT) on the original image and then entropy coding the quantized DCT coefficients. WebP [22] is a more recent image compression standard commonly used on the web. Both the basic JPEG standard and WebP are *non-progressive* in that they require the reception of an entire compressed image to reconstruct it effectively. In contrast, to handle unpredictable bandwidth, JPEG also provides a *progressive* mode based on spectral selection and successive approximation in the entropy coding procedure. It allows images to be reconstructed based on an arbitrary amount of encoded data, with more data resulting in more accurate reconstruction.

B. Neural Compression for Image Reconstruction

Recent works in image compression explored deep learning approaches using AEs to learn latent representations of images. Early efforts on such neural compression techniques [23]–[25] are non-progressive because their neural network architecture cannot handle partially received data. Recent efforts are to

support *progressive* encoding in neural compression. Lu *et al.* [26] propose nested quantization with multiple scaling levels that refines all latents progressively. Toderici *et al.* [27], [28] exploit recurrent neural networks (RNN) to support progressive image compression, where a multi-iteration compression architecture is proposed to iteratively compress the residual signals between the input and reconstructed image patch. While neural compression has shown advantages over traditional compression approaches in compression ratio, a disadvantage of existing deep models for image compression is their complexity and computation cost, often prohibitive for resource-constrained IoT devices. It is also important to note that the aforementioned neural compression approaches are designed to optimize image reconstruction quality instead of application-specific inference performance such as image classification accuracy.

C. Neural Compression for Inference

Recent advances in neural compression target optimizing inference performance. In the context of edge computing, several works propose approaches where the device and the edge server work collaboratively to complete the inference task [29], [30]. In this paradigm, a DNN model for inference is split into two separate parts, a head model and a tail model, which are deployed at the device and the edge server, respectively. To compress the intermediate features transmitted from the device to the edge server, BottleNet and its extensions [9], [12] inject a bottleneck AE at the split point of the original model. To improve encoding efficiency, Yao *et al.* [10] propose an asymmetric AE structure to lower the encoding overhead at the local device. To further reduce redundant computation, Matsubara *et al.* [11], [31] apply knowledge distillation to train a lightweight head model with an embedded bottleneck that compresses the input image while performing part of the inference. In subsequent work [13], this head distillation framework was extended to achieve an explicit learnable trade-off between compression ratio and reconstruction distortion by combining neural image compression techniques [24], [32], [33]. However, despite the significant advances in neural compression for inference, none of these solutions support progressive compression, which is essential in adapting to variable bandwidth, as is common in wireless sensor networks.

A recent neural compression approach, Starfish (Hu *et al.*) [34], accounts for data loss during transmission by introducing random dropouts at the bottleneck of their proposed AE to foster resiliency against such losses. However, as Starfish does not differentiate or prioritize features in terms of their contributions to inference accuracy, it is not designed to optimize classification performance under different deadlines and available bandwidth.

Our proposed solution, PNC, is specifically designed to optimize image classification accuracy in the context of edge offloading under timing constraints. In contrast to existing solutions, PNC can adapt to variable bandwidth through progressive encoding. It lets the edge server perform effective image classification based on the subset of encoded features

that have been received by the offloading deadline. In addition, to minimize computational load on IoT devices, PNC employs a lightweight rateless AE and does not require inference to be partially executed on the devices.

IV. PROBLEM FORMULATION

We consider an edge image classification task where images are captured in a local device and must be transmitted to an edge server for classification purposes. Because of bandwidth limitations, images are encoded (compressed) before transmission and subsequently decoded at the edge server before classification. Our goal is to achieve the highest classification accuracy possible at the edge server, given the bandwidth constraints of the network and the image offloading deadline. We focus on the timing constraint for image offloading subject to variable bandwidth.

Assume a sequence of images, \mathbf{x}_i , $i = 1, 2, 3, \dots$, captured by the local device at fixed time intervals of period T . Assume the first image is captured at t_1 , then \mathbf{x}_i will be captured at $t_i = t_1 + (i - 1)T$. In this work, we consider a common scenario where the image offloading must be completed before the next image is encoded, i.e., the deadline for offloading an image \mathbf{x}_i is the arrival of the encoded features of the next image \mathbf{x}_{i+1} .

Specifically, we split the process into three steps: On-device encoding and network offloading, and subsequently edge decoding and classification.

On-device Encoding. Upon the arrival of an image \mathbf{x}_i , the device encodes and sends the data to a remote server for image classification. Given an encoder, f_θ , the encoded data for each image is in the form of a byte array that can be expressed as, $\mathbf{z}_i = f_\theta(\mathbf{x}_i)$. The encoding latency for \mathbf{x}_i is denoted as t_i^f .

Network Offloading. We assume that the transmission of image \mathbf{x}_i can continue until the next image \mathbf{x}_{i+1} is encoded and ready for transmission, so that its deadline is $t_d = t_i + T + t_{i+1}^f$. As the network bandwidth, $b(t)$, varies with time, the encoded data size S_i that can be transmitted for image \mathbf{x}_i by its deadline is given by $S_i = \int_{t_i+t_i^f}^{t_i+T+t_{i+1}^f} b(t)dt$.

Denote $\text{truncate}(\mathbf{z}_i, S_i)$ as the data when truncating the end of byte array \mathbf{z}_i to size S_i , such that $\text{size}(\text{truncate}(\mathbf{z}_i, S_i)) \leq S_i$ (note that when $\text{size}(\mathbf{z}_i) \leq S_i$, $\text{truncate}(\mathbf{z}_i, S_i) = \mathbf{z}_i$). The actual data, $\tilde{\mathbf{z}}_i$, that the client can send within the deadline is,

$$\tilde{\mathbf{z}}_i = \begin{cases} \mathbf{z}_i, & \text{if } \text{size}(\mathbf{z}_i) \leq S_i, \\ \text{truncate}(\mathbf{z}_i, S_i), & \text{otherwise.} \end{cases} \quad (1)$$

Decoding and Classification. When the edge server receives the (potentially truncated) encoded data, $\tilde{\mathbf{z}}_i$, the decoder, g_ϕ , will decode the data, $\mathbf{y}_i = g_\phi(\tilde{\mathbf{z}}_i)$. Assume that h is some image classifier and a_i is the output of the image classifier, i.e., the probability distributions among the possible classes with input \mathbf{y}_i , $a_i = h(\mathbf{y}_i) = h(g_\phi(\tilde{\mathbf{z}}_i))$. We can evaluate the accuracy of the prediction with the commonly used performance metric *top-n accuracy*, R , of the image classification task on image \mathbf{x}_i with ground-truth class of gt_i as,

Offline Training Phase

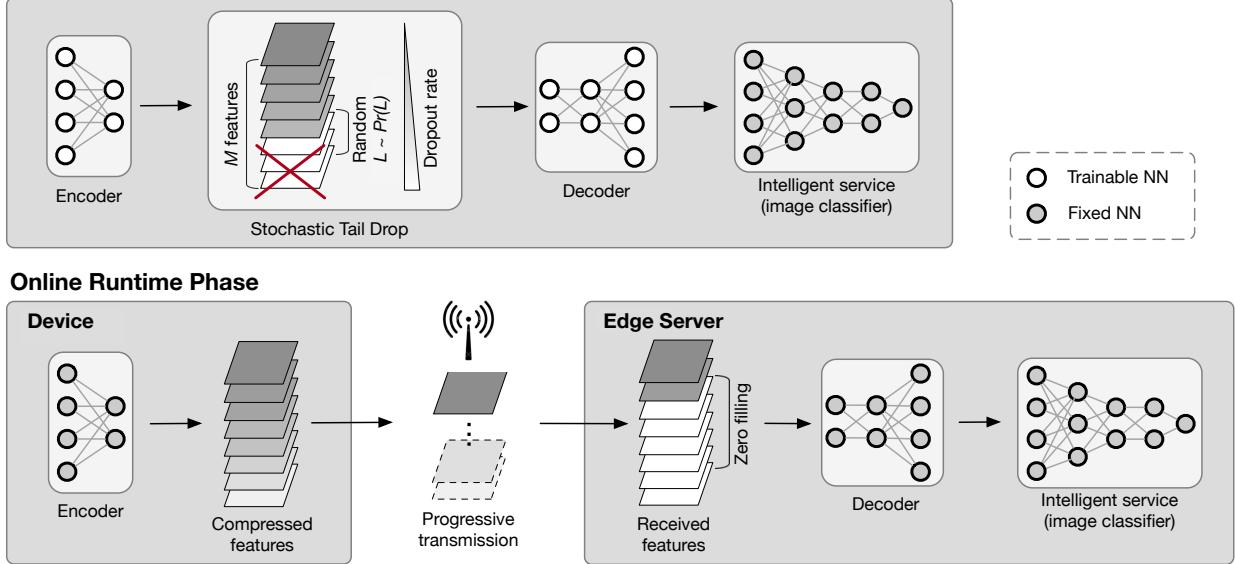


Figure 1: System overview of our PNC offloading framework.

$$R(a_i, gt_i) = \begin{cases} 1, & \text{if } gt_i \in \{n \text{ most likely predictions}\}, \\ 0, & \text{if } gt_i \notin \{n \text{ most likely predictions}\}. \end{cases} \quad (2)$$

Our goal is to design a flexible configuration for edge offloading and optimize the learnable parameters of the image encoder and decoder, θ and ϕ , that maximize the accuracy of the image classification task, under variable bandwidth $b(t)$ and the constraint of the deadline imposed by T_d .

In summary, our goal is to design an offloading solution that can achieve high accuracy with various combinations of time-varying bandwidth and deadline, *i.e.*, an unknown amount of transmitted data in an adaptive fashion. The encoder should capture important features for classification and help the client progressively transmit the most important features first, followed by other, less important features. The decoder should handle incomplete data arriving at the edge server and decode an input for the image classifier to achieve the best classification accuracy given the amount of data received.

V. DESIGN OF PNC

This section introduces the design of our PNC framework for progressive neural compression and adaptive offloading. We target a typical real-time edge computing architecture comprising embedded IoT devices and an edge server connected by a wireless sensor network, where image offloading is regulated by a deadline constraint. The key idea of our framework is to formulate the problem of progressive offloading as training a rateless compressive AE that serves multiple objectives, where each objective corresponds to optimizing the inference performance using a certain amount of data partially received during transmission. We exploit recent advances in dimensionality reduction for multi-rate image compression and extend such a

concept to support progressive offloading of inputs for edge-assisted image classification, thereby allowing the system to adapt to varying network bandwidth under timing constraints.

A. System Overview

Fig. 1 illustrates the offline training process and the end-to-end image classification pipeline featured by the PNC framework. During the offline training phase, the PNC trains a rateless compressive AE by dropping a random number of encoded features (by filling these features with zeros) at the tail of its bottleneck during each training iteration to emulate the reception of incomplete image data (Section V-B and V-C). When the PNC is later deployed at runtime, the device encodes the image into a fixed number of features and transmits to the edge server as many features as possible within the time limit. With the features that failed to be transmitted in time replaced with zeros, the edge server decodes the received data and predicts the class of the input image (Section V-D).

B. Multi-objective Neural Compression

1) *AE-based Fixed-rate Compression*: A vanilla compressive AE consists of an encoder and decoder. The encoder, denoted as $\mathbf{z} = f_\theta(\mathbf{x})$, maps the input image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ (H , W , and C define the size and number of channels of the input. For RGB images, $C = 3$) to a lower-dimensional latent representation $\mathbf{z} \in \mathbb{R}^{h \times w \times M}$ (h and w define the size of the latent feature map for each channel, M is the number of latent channels). θ is the parameters of the encoder. The decoder, denoted as $\hat{\mathbf{x}} = g_\phi(\mathbf{z})$, aims to reconstruct the original image \mathbf{x} by mapping the latent representations back to the data space. ϕ is the parameters of the decoder. The aim is to find θ, ϕ that jointly minimize a loss:

$$\theta, \phi = \arg \min_{\theta, \phi} \mathcal{L}(\theta, \phi) \quad (3)$$

with $\mathcal{L}(\theta, \phi) = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \|\mathbf{x} - g_\phi(f_\theta(\mathbf{x}))\|^2$ optimizing the AE to reconstruct the input images and minimize the distortion [23]–[25]. In compressive offloading [10], [11], however, the AE is often trained to optimize inference performance (e.g., classification accuracy). Here the loss \mathcal{L} can be generalized to inference losses (e.g., cross entropy) to meet the needs of applications. The natural property of dimensionality reduction of AE enables its potential of being used in compressive offloading for edge computing [10], [11], [13] where \mathbf{z} is transmitted as compressed data to improve efficiency.

2) *Multi-objective Optimization for Rateless Compression*: The problem with using a standard AE for compressive offloading is that the compression rate remains fixed due to the fixed-sized bottleneck \mathbf{z} . In wireless sensor networks with varying bandwidths and offloading deadlines, the bottleneck features may not be fully received. This leads to significant distortion and performance decline for the target deep learning service, as the bottleneck features are always complete during training.

To reduce the distortion and optimize classification performance under the situation where only a subset of the bottleneck features are successfully received by the edge, *i.e.*, the first K features ($K \leq M$), we can incorporate the objective w.r.t. reconstructing data using the top K channels,

$$\mathcal{L}(\theta, \phi, K) = l(\mathbf{x}, g_\phi(\text{Concat}[\mathbf{z}_{[1:K]}; \mathbf{0}])) \quad (4)$$

where l represents the choice of objective function to optimize during the two-stage offline training process, which we will discuss in Section V-C, $\text{Concat}([\cdot])$ denotes tensor concatenation, and $\mathbf{z}_{[1:K]}$ denotes the first K latent feature channels of \mathbf{z} . To consider all possible K values under different bandwidth, we can train a AE (f_θ, g_ϕ) by optimizing the following *multi-objective* problem:

$$\min_{\theta, \phi} \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} [\mathcal{L}(\theta, \phi; 1), \mathcal{L}(\theta, \phi; 2), \dots, \mathcal{L}(\theta, \phi; M)] \quad (5)$$

However, there are no close-form solutions to Eq. (5). Since each objective is not independent of each other (*i.e.*, sharing the model parameters θ and ϕ), the rateless AE must account for the trade-off among various bottleneck dimensionality to reach the *Pareto frontier* solutions as close as possible.

Recent work [16] proposed the *stochastic taildrop* regularization as an iterative approximation to the solutions of Eq. (5) for flexible neural dimensionality reduction. As illustrated in the “Offline Training Phase” diagram in Fig. 1, in each training step, we randomly drop the last L latent channels ($L = M - K$) of the bottleneck by replacing them with 0 during training. Intuitively, this is equivalent to alternatively optimizing each of the objectives with different K at different iterations. On top of *stochastic taildrop*, we added additional gradient control over the iterations for more stable training. The iterative joint optimization is shown in Algorithm 1. At each training iteration (the innermost for loop), we randomly draw the length of dropped tail L from a distribution [16] (e.g., uniform) and calculate the gradient $\mathbf{g}_\theta^{(m)}, \mathbf{g}_\phi^{(m)}$ following the objective \mathcal{L} . After every M iterations, the training process

Algorithm 1: Iterative optimization of Eq. (5) with stochastic taildrop.

Data: Number of epochs N_e , number of batches N_b , mini-batch image set $\mathcal{X}_{train} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_{N_b}\}$, predefined distribution $Pr(L)$, bottleneck width of the AE M

Result: Learnable parameters of the encoder, θ , and the decoder, ϕ .

```

for  $1 \leq n_e \leq N_e$  do
  for  $1 \leq n_b \leq N_b$  do
    for  $1 \leq m \leq M$  do
       $L \sim Pr(L)$ ; /* Length of dropped tail. */
       $K \leftarrow M - L$ ;
      Gradient  $\mathbf{g}_\theta^{(m)}, \mathbf{g}_\phi^{(m)} \leftarrow \nabla_{\theta, \phi} \mathbb{E}_{\mathbf{x} \sim \mathbf{X}_{n_b}} \mathcal{L}(\theta, \phi; K)$ ;
    end
     $\bar{\mathbf{g}}_\theta, \bar{\mathbf{g}}_\phi \leftarrow \frac{1}{M} \sum_{m=1}^M \mathbf{g}_\theta^{(m)}, \frac{1}{M} \sum_{m=1}^M \mathbf{g}_\phi^{(m)}$ ;
    Update  $\theta, \phi$  using gradient  $\bar{\mathbf{g}}_\theta, \bar{\mathbf{g}}_\phi$  on batch  $\mathbf{X}_{n_b}$ .
  end
end

```

will calculate the average gradient and update the learnable parameters, θ and ϕ .

C. Two-stage Offline Training

In this section, we introduce how to optimize our rateless AE for downstream inference tasks (e.g., image classification) via a two-stage training method.

1) Unsupervised Pretraining for Image Reconstruction:

We first pre-train the rateless AE by optimizing Eq. (5) via Algorithm 1 on reconstructing input images $\mathbf{x} \sim \mathcal{X}$, in which

$$\mathcal{L}(\theta, \phi, K) = \left\| \mathbf{x} - g_\phi \left(\text{Concat} [f_\theta(\mathbf{x})_{[1:K]}; \mathbf{0}] \right) \right\|^2 \quad (6)$$

The unsupervised pretraining helps prepare the AE parameters θ, ϕ to a generalizable state that adapts to any potential target machine learning (ML) service.

2) Knowledge Distillation for Inference:

The parameters θ, ϕ trained by the above unsupervised stage are not optimized for inference performance (e.g., classification accuracy). In order to achieve a better performance-efficiency tradeoff for inference, the pretrained rateless AE needs to further distill knowledge from the target inference task. Knowledge distillation (KD) was first proposed in [35] and widely used by [10], [13], [31] and showed improved inference performance than training from scratch. In KD, a (usually smaller) student model is trained to produce similar inference output as the (usually much larger) target teacher model. In this paper, we consider a deep image classifier to instantiate $h(\mathbf{x})$ and extend the scope of stochastic taildrop from image compression and reconstruction to image classification. Assume the deep model of the target inference task is $h(\mathbf{x})$. We use KD to fine-tune the rateless AE to reconstruct input data that optimize the inference performance with the new objective function specifically for the classification task,

$$\mathcal{L}(\theta, \phi, K) = \text{CE} \left(h(\mathbf{x}), h \left(g_\phi \left(\text{Concat} [f_\theta(\mathbf{x})_{[1:K]}; \mathbf{0}] \right) \right) \right) \quad (7)$$

Here $\text{CE}(\cdot, \cdot)$ denotes cross-entropy loss as used in [35]. During the KD training, the target model $h(\mathbf{x})$ is treated as a black box with the parameters frozen, and only the AE parameters, θ and ϕ , are updated.

D. Progressive Compression for Adaptive Offloading

With the stochastic taildrop regularization and iterative optimization (Algorithm 1) during training, the p^{th} feature $\mathbf{z}_{[p]}$ is preserved and used for reconstruction more often than any succeeding feature $\mathbf{z}_{[q]} (1 \leq p < q \leq M)$. Intuitively, the AE will be more robust to the absence of $\mathbf{z}_{[q]}$, so features $\mathbf{z}_{[1:M]}$ are given decreasing importance to the target inference task performance.

As shown in the diagram of ‘‘Online Runtime Phase’’ in Fig. 1, we design an end-to-end image classification system to support progressive neural compression and image offloading over wireless sensor networks. After encoding original images into a set of compressed features, we progressively transmit the features from the most important $\mathbf{z}_{[1]}$ to the least important $\mathbf{z}_{[M]}$ in descending order to the edge server for further reconstruction and inference. Given limited current bandwidth and a time constraint, the network may be able to transmit only a subset of the features (i.e., top K') to the edge server before the deadline. Then the received most important K' features will be used for data reconstruction and inference (remaining features filled with 0). These received features are exactly the optimal set of features that maximizes the inference performance of the image under the constraint of sending only K' features. In this manner, the trained rateless AE achieves progressively better inference performance with more features offloaded. This progressive neural compression framework enables our compressive AE to adapt flexibly to variable network bandwidth while achieving better average inference performance.

E. Quantization and Huffman Coding

To further reduce the data size of intermediate features \mathbf{z} that need to be transmitted to the edge server, we apply additional optimizations, including model and feature quantization and Huffman coding [36] before data transmission.

Model Quantization. Model quantization [37], [38] converts floating point operations and data in ML models to quantized ones, usually in the form of 8-bit integers. We use the automated TensorFlow Lite (TFLite) converter, an official tool by TensorFlow, for such a purpose. The conversion speeds up the encoding on resource-constrained IoT devices and helps reduce the output data size of the ML model.

Feature Quantization. We empirically find that even after model quantization to 8-bit integers (0-255), there is still abundant room for more aggressive quantization. After encoding, we further quantize the data to 64 values, represented by 6 bits, spaced evenly between 0 and 255. We select the number 64 because it has a relatively small impact on the classification accuracy while directly reducing the data size by 25% if we use 6-bit binary coding.

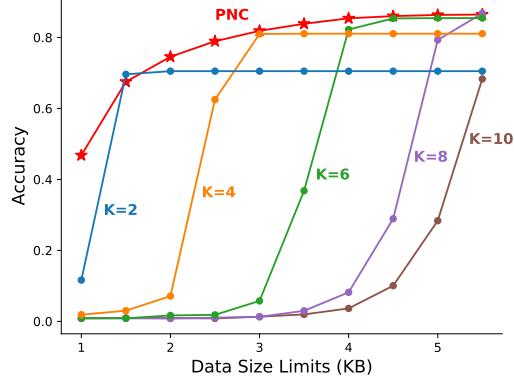


Figure 2: Classification accuracy vs data size limits of PNC and various fixed rate AEs. PNC is in red. For fixed-rate AEs, a label with the number of latent channels, K , is next to each curve with the corresponding color.

Feature-wise Huffman Coding. We find that the data distributions within each feature are different. For the best compression result, we train M different Huffman tables for features at different positions. During the online runtime phase, each quantized feature will be further compressed individually with the corresponding Huffman table.

F. Progressive Property of PNC

In this section, we provide insights into how the design of PNC facilitates its progressiveness and adaptability to bandwidth variations with little impact on its classification performance. To better demonstrate the effect of taildrop on multi-objective rateless compression, we remove tail-drop regularization from PNC and vary the number of latent channels, M , in the model to create multiple *fixed-rate AEs*. The bottleneck sizes of these fixed-rate AEs are $32 \times 32 \times K$, where $K \in \{2, 4, 6, 8, 10\}$ represents the latent channel numbers and can effectively translate to various encoding rates. All fixed-rate AEs are trained and optimized following the same strategy as PNC, as discussed in Section V-C and Section V-D. As a result, each latent channel will be of size 1KB after model quantization and become 0.75KB after feature quantization. Huffman coding will further improve coding efficiency and reduce the average channel size. We utilize the ImageNet validation set with 35,000 images for training, 5,000 for validation, and 10,000 for the following simulation evaluation.

Fig. 2 shows, as a function of data size limitation, the classification accuracy of PNC and those fixed-rate AEs. First, we note that PNC consistently delivers classification accuracy close to the *best* performance of each fixed-rate AE (even though the fixed-rate AE is optimized for the target size of latent features). The slight drop in performance in comparison to the best fixed-rate AE at each data size is attributed to the constraints imposed by stochastic taildrop during training (e.g., the accuracy for $K = 2$ is 69.62%, just slightly higher than PNC’s 67.51% at 1.5KB. The accuracy for $K = 8$ is

86.79%, barely above PNC’s 86.46% at 5.5KB.). Overall, the multi-objective design enabled by stochastic taildrop during training allows PNC to balance the inference performance under multiple compression rates during inference. PNC is therefore able to achieve progressive encoding at a negligible cost in classification accuracy across a wide range of data sizes. In other words, the accuracy of PNC gracefully degrades as the number of offloaded features decreases.

In contrast, the performance of fixed-rate AEs decreases rapidly as the data size limit drops substantially from their respective design targets, indicating a failure of the decoder to receive the complete set of features. Without stochastic taildrop, each feature in a fixed-rate AE will be equally important and indispensable for decoding and classification. This becomes a serious problem for DNNs-based image encoders with fixed rates when the amount of data sent to the server is unknown or varies. Similarly, selecting the best model that guarantees the transmission of all encoded data within a time limit becomes challenging. Therefore, a significant advantage of PNC lies in its progressive encoding that enables the system to adapt to unpredictable bandwidth and different offloading deadlines while maintaining classification accuracy.

G. Image Offloading Architecture

We design a distributed system architecture (Fig. 3) for end-to-end image classification comprising multiple stages: image encoding and offloading on the IoT device (client), followed by decoding and classification on the edge server. Recall our design targets a common application scenario where the image classification task is triggered by images captured periodically. We focus on the latency of the image offloading stage subject to varying bandwidth. In our design, the image offloading stage has a time constraint that the transmission of features belonging to an image must end before features of the next image are generated.

To improve resource utilization, our system employs multiple threads to pipeline the stages of the end-to-end processing. Specifically, image encoding is released in response to the arrival of a new image. Thread TH_{encode} first fetches and encodes the image into compressed features. Thread TH_{send} then sends the features to the edge server over the network. To provide fine-grained control, TH_{send} divides the features of an image into small, 64-byte data blocks and sends them sequentially. Before sending each block, TH_{send} will check whether the encoded data of the next image are ready. If the transmission of img_n features is not finished before the first feature of img_{n+1} is encoded, TH_{send} terminates the transmission, appends a stop signal to the data stream indicating the termination of transmission for img_n , and proceeds to transmit img_{n+1} .

PNC uses Huffman encoding as the final optimization step after an image is compressed into features. To reduce the latency overhead caused by the Huffman encoding, TH_{send} is invoked to offload the first feature once it is encoded, while the remaining features are encoded with the Huffman table in parallel using a third thread (not shown in the figure).

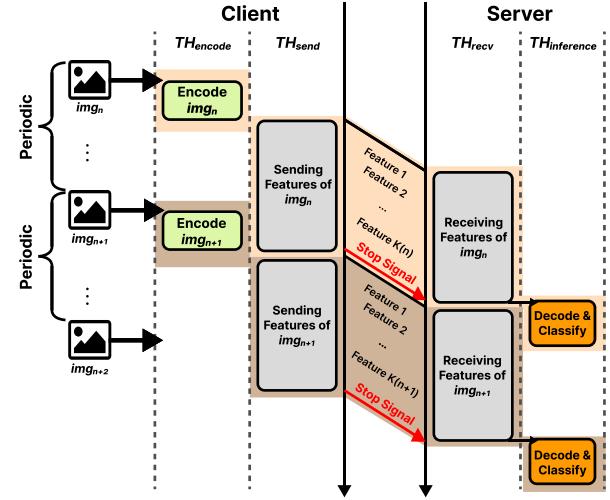


Figure 3: The image offloading architecture in our end-to-end image classification system.

The edge server runs two threads: Thread TH_{recv} receives the bytes from the network interface until it receives a complete image or the stop signal. Then thread $TH_{inference}$ decodes and classifies img_n using the received data. Our architecture is implemented in Linux-based OSs on both the device and the server, although, in principle, it may be implemented in other OSs supporting multiple threads.

VI. IMPLEMENTATION

We implement an end-to-end image classification system comprising an IoT device and an edge server. The system employs PNC for image offloading over a low-power wireless sensor network.

A. Hardware Platform

We use a Raspberry Pi 4 Model B with Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, 8GB RAM, and Raspberry Pi OS 10 (Kernel version: 5.10.63) as the embedded IoT device and a desktop with Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz, Nvidia GeForce RTX 3090 GPU, 64GB RAM and Ubuntu 20.04.1 (Kernel version: 5.15.0-69-generic) as the edge server. The device and the server communicate with each other using IEEE 802.15.4 radios (nRF52840 development kits “nRF52840 DK” produced by Nordic Semiconductor). The kit offers a serial port interface that allows the client and the server to transmit and receive data streams with the IEEE 802.15.4 radio at the application level. The MAC layer of the radio uses re-transmissions to overcome transmission failures. As a result, the available network bandwidth drops in the presence of external inference and packet losses.

B. Design Choice of Autoencoder

We implement the AE in PNC with TensorFlow². The details of our compressive AE are shown in Fig. 4. Inspired by Yao *et al.* [10], we adopt an asymmetric design of the AE

²<https://www.tensorflow.org>

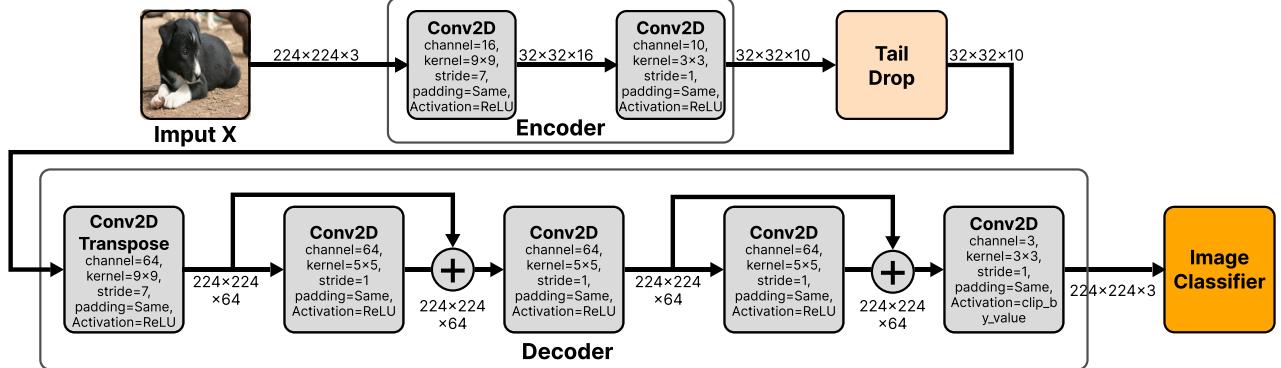


Figure 4: The AE architecture used in PNC. The encoder contains two 2D convolution layers. The decoder contains five layers. The input/output size of each layer is labeled between layers.

to reduce the encoding overhead at the device. The encoder and decoder contain two and five layers, respectively. The encoder compresses the input image into 10 features of size 32×32 . We make the encoder lightweight, *i.e.*, with only two 2D-Convolution (Conv2D) layers to match the computational capacity of the embedded device. We use clip-by-value as the activation function in the last layer to limit the AE output within a reasonable range. Empirically, we find that PNC trained with clip-by-value achieves higher classification performance than using the common sigmoid or tanh activation functions. For tail length selection, we use the uniform distribution following [16], *i.e.*, $L \sim \text{Uniform}(0, M - 1)$, and zero out the last L channels during training.

We train the AE in PNC following the steps in Section V-C. We use a learning rate of 0.001 for the first stage and 0.005 for the second. We set the batch size to 4 and the number of epochs to 70, and use the Adam optimizer during both training stages.

We use ImageNet³ for model training and system evaluation. We use the off-the-shelf EfficientNet-B0 [39] pre-trained on the ImageNet training set as our image classifier. Since the training set is already used to optimize the image classifier, we extract additional training data from the ImageNet validation set to train PNC’s AE. We assigned 35,000 images for training, 5,000 for validation, and 2,000 for testbed experiments. We also resize all the images to $224 \times 224 \times 3$ to match the input size of the classifier.

VII. EVALUATION

In this section, we compare the encoding efficiency and classification performance of PNC against a set of traditional and state-of-the-art baselines under different timing constraints (deadlines) and network conditions.

A. Baselines for Comparison

We compare PNC against a wide range of baselines, including traditional and neural compression approaches, as well as

non-progressive and progressive encoding schemes of different levels of encoding complexity.

- **WebP:** WebP is a state-of-the-art, non-progressive, traditional encoder. In addition to the quality factor, q , that controls the reconstruction quality of the image, WebP has another parameter, *method*, which is an integer ranging from 0 to 6, indicating the quality-speed trade-off. Method 0 has the fastest encoding and method 6 has the best quality and larger encoding overhead. We select *method* = 6 to show the best compression performance and select a range of quality factors to evaluate its performance.
- **Progressive JPEG:** Progressive JPEG is a progressive variant of JPEG that supports image decoding with partial image data. We explore the quality factor empirically and report the results with $q = 30$, which balances classification accuracy and compression ratio.
- **RNN-TFLite:** We include a recurrent neural network (RNN) method [28] as a state-of-the-art method for progressive neural compression. The model is trained with the same set of training images as our model. We benchmark the encoding latency of RNN TFLite, a quantized version of the original RNN model in TensorFlow Lite. With an image of size 224×224 , this method generates 784 bytes of data during each iteration. Note that the original work proposes multiple network implementations. We adopt a representative one with long short-term memory (LSTM) as the recurrent unit and “additive reconstruction” as the image reconstruction approach from our decoder’s output. We select the implementation without entropy coding to mitigate the complexity of the model. Regardless, as shown in Section VII-B, RNN methods remain prohibitively expensive for embedded devices.
- **Starfish** [34]: Starfish is a state-of-the-art AE-based approach designed to tolerate data loss. During training, Starfish utilizes a dropout layer located at the bottleneck of the AE to make the system resilient to data loss. Since the original Starfish was evaluated on different datasets, we retrained Starfish with our training dataset and optimized the

³<https://www.image-net.org/>.

Table I: Comparison of encoding overhead. For different WebP methods, m represents the *method*. For all applicable methods, q represents the quality factor.

Method	Configuration	Average Encoding Overhead (ms)
PNC	#thread = 1	11.8
	#thread = 2	11.4
WebP	$m = 0, q = 0$	6.0
	$m = 0, q = 20$	6.7
	$m = 6, q = 0$	32.5
Prog. JPEG	$m = 6, q = 20$	48.6
	$q = 30$	5.8
Starfish	One Patch	62.9
RNN-TFLite	One Iteration	1900

model with quantization and Huffman coding. Note that, to reduce AE complexity and encoding latency, Starfish evenly divide images into four patches and encode them individually.

B. Encoding Overhead

For each image compression method, we measure the image encoding overhead on the embedded device (Raspberry Pi 4). As shown in Table I, PNC takes on average 9.8ms to encode the image into features of sorted importance. Additional “feature quantization” and “Huffman encoding” take an additional 2.0ms. The average encoding overhead is hence $9.8 + 2.0 = 11.8$ ms⁴. WebP encodes an image in 6.0ms with *method* = 0, but the overhead grows significantly to over 32.5ms for *method* = 6, much longer than PNC. Progressive JPEG incurs an average overhead of 5.8ms.

In contrast, the neural compression baselines have much higher overheads than PNC and the traditional compression methods. Starfish takes an AE with larger numbers of intermediate layers and channels than PNC, resulting in a higher encoding overhead of 62.9ms for only the first image patch (out of a total of four), 51.1ms longer than PNC. The TFLite-optimized RNN baseline (RNN-TFLite) takes on average 1.9s to encode an image for one iteration when executing with 4 threads and takes 3.0s with only 1 thread. Hence, the prohibitively high complexity of RNN models makes them unsuitable for embedded devices.

C. Performance under Different Deadlines and Bandwidth

1) *Experimental Setups*: We deploy our image classification system in a multi-room indoor space as shown in Fig. 5. The system comprises the embedded device, the jammer, and the edge server described in Section VI-A. The server and the device are placed in Room 1 and Room 2, respectively, as shown in Fig. 5. The device offloads encoded images to the server over an IEEE 802.15.4 wireless link. The jammer (a Raspberry Pi 3 Model B) is placed close to the client and generates traffic at different rates using *iperf3*.

⁴The average encoding overhead can be reduced to 11.4ms if two threads are used to run the TFLite inference for the encoder. However, since the JPEG and WebP implementations in the Python image library, Pillow, used in our experiments, only support single-threaded encoding/decoding, for fairness of comparison, we also kept PNC single-threaded in our experiments.

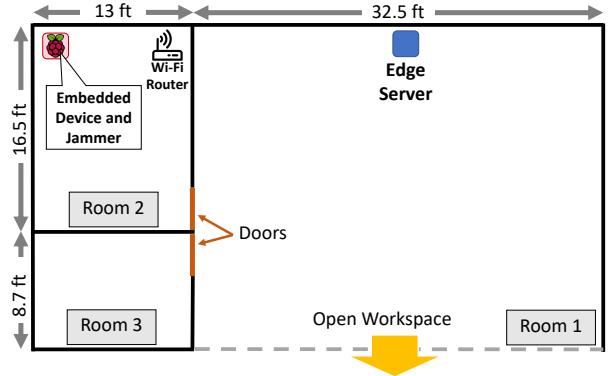


Figure 5: Layout of the indoor space used for the experiments. To minimize interferences from other environmental factors, all experiments are conducted at night when no one is present.

It communicates with a Wi-Fi router on a 2.4GHz Wi-Fi channel that overlaps with the IEEE 802.15.4 channel used for image offloading. Jamming intensity is varied by controlling *iperf3*’s transmission rate.

We conduct the experiments under (1) different network scenarios and (2) different timing constraints. We create three network scenarios with different levels of wireless interference and, consequently, different bandwidths between the device and the server:

- **Scenario 1 (No Jamming)**: There is no jamming traffic.
- **Scenario 2 (Light Jamming)**: *iperf3* on the jammer generates traffic at a rate of 10 Mbps.
- **Scenario 3 (Heavy Jamming)**: *iperf3* on the jammer generates traffic at a rate of 20 Mbps.

For each network scenario, we repeat the experiments with different image arrival periods (T) of 300, 500, and 700 ms (and hence image offloading deadlines of approximately the same values).

We conduct experiments on our testbed with a fixed set of 2,000 images drawn from the ImageNet validation set. As our system terminates the offloading of an image upon the arrival of the first feature of the next image, all the timing constraints for image offloading are met in the experiments. Henceforth, we evaluate the classification performance of PNC and the baselines under different network scenarios and timing constraints. We use a standard metric, *top-n accuracy*, which is commonly used in previous edge computing works [10], [40], [41], to quantify classification performance. Specifically, as per Eq. (2) for $n = 5$, top-5 accuracy assigns a score (reward) of 1 for the received image if its ground-truth class is among the 5 most likely classes produced by the classifier and 0 otherwise.

Fig. 6 shows the top-5 accuracy of PNC and the baselines including the non-progressive baseline, WebP (*method* = 6), the progressive baseline, Progressive JPEG (with the best configuration, $q = 30$), and the neural-compression baseline, Starfish. We leave RNN out due to its prohibitive encoding overhead on embedded devices (see Section VII-B). To further analyze the behavior of different solutions, we also show the

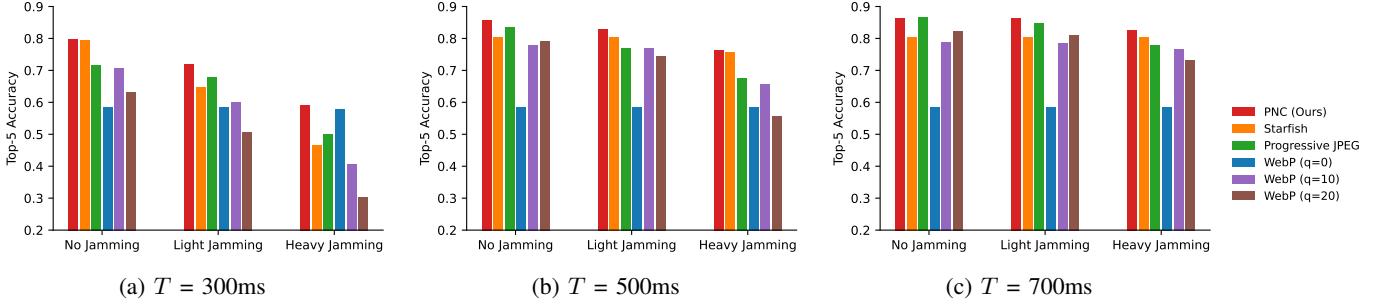


Figure 6: Classification accuracy. Each subfigure shows the results in a network scenario with different image arrival periods.

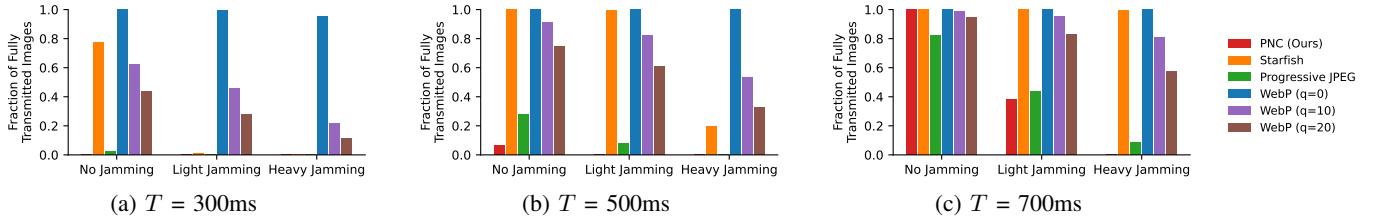


Figure 7: Fraction of images completely offloaded within the time limit. Each subfigure shows the results in a network scenario with different image arrival periods.

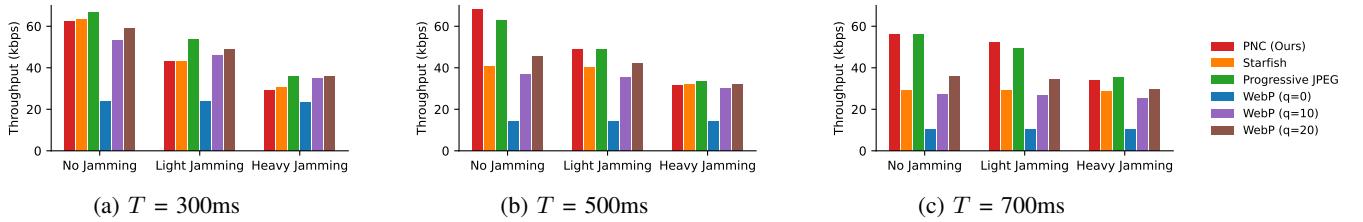


Figure 8: Network throughput during image offloading. Each subfigure shows the results in a network scenario with different image arrival periods.

fractions of images fully offloaded within the time constraints (Fig. 7) and the network throughput during image offloading (Fig. 8).

2) *Classification Performance*: As expected, the image arrival period (T) and the available network bandwidth both affect the amount of data that can be transmitted to the server and, therefore, the methods' classification accuracy, *i.e.*, lower accuracy when T decreases, or network jamming worsens.

We first look at the performance of **WebP** with different quality levels ($q = 0, 10, 20$). As a traditional compression approach not optimized for inference, WebP generally delivered poor classification performance. Moreover, as a non-progressive approach, WebP cannot adapt to different deadlines and network conditions. For instance, WebP with the *highest* quality ($q = 20$) achieved the highest classification accuracy among the WebP variants with $T=700\text{ms}$ and no jamming (Fig. 6c). In contrast, WebP with the *lowest* quality ($q = 0$) achieved the highest classification accuracy among WebP variants with $T=300\text{ms}$ and heavy jamming (Fig. 6a). This is consistent with the fact that WebP ($q = 20$) is able

to offload most of the high-quality images fully in the former case, as allowed by the longer deadline and high bandwidth. In contrast, WebP ($q = 20$) can offload only about 10% of the images fully in the latter case. On the other hand, while WebP ($q = 0$) is able to fully offload almost all the images in all the testing scenarios, its low quality limits the classification accuracy. This is consistent with the low throughput of WebP ($q = 0$) in all the experiments (Fig. 8), which shows WebP ($q = 0$) is not able to utilize the bandwidth available due to its aggressive compression at the cost of quality. We note that all variants of WebP generally achieve low throughput in comparison to PNC and the other baselines due to the small data sizes produced by WebP encoding.

We then analyze the progressive baseline, **Progressive JPEG**. In contrast to WebP, Progressive JPEG can effectively adapt to deadlines and network conditions. As shown in Fig. 6, with Progressive JPEG, classification accuracy improves with longer deadlines and less network interference, which allows Progressive JPEG to offload more data. This is consistent with Fig. 8, which shows the system achieves

higher throughput with Progressive JPEG in the presence of lower interference. Interestingly, the system achieves its classification performance even though only small fractions of the images are fully offloaded with Progressive JPEG (Fig. 7), thanks to its progressive encoding. However, Progressive JPEG consistently underperforms compared to PNC, which suggests the strength of neural compression optimized for inference. The only scenario when Progressive JPEG and PNC achieve similar classification accuracy is when $T=700$ ms with no jamming. Given the long deadline and large bandwidth in this scenario, PNC successfully transmits all the features associated with each image by the deadline, while Progressive JPEG manages to refine many of the images to the point where the reconstructed image is sufficient for accurate image classification (Fig. 8).

The neural compression baseline, **Starfish**, is designed to tolerate data loss or incomplete data. As a result, it delivers relatively robust performance across different scenarios. However, Starfish has two limitations. First, it does not differentiate features when generating and transmitting them. When the deadline and bandwidth are severely limited, e.g., $T=300$ ms with heavy jamming, it suffers a significant loss in accuracy as the features offloaded are not sufficient for classification. In contrast, through its tail-drop strategy, PNC generates features with different importance for classification and sends the features in order of importance. As a result, PNC manages to offload the most important features within the timing constraint, even in the presence of heavy jamming. This ensures that PNC consistently outperforms Starfish including in scenarios with tight timing constraints and low bandwidth. In addition, Starfish cannot take advantage of situations when a larger amount of data are allowed to be transmitted to the server, *i.e.*, with $T=700$ ms. This is because the Starfish encoder produces a relatively small amount of data (on average 2.53KB) after compression, which in turn upper-bounds the amount of data that can be offloaded. This is consistent in the fact Starfish offloads most of the images fully with $T=700$ ms (Fig. 7c) while showing low network throughput (Fig. 8c) because it does not generate enough data to fully utilize the bandwidth available.

PNC consistently outperforms all the baselines across all the scenarios in our experiments. Like Progressive JPEG, PNC maintains robust classification performance (Fig. 6) while offloading only small fractions of images fully to the server (Fig. 8). The consistent performance is attributed to its novel strategy of progressively transmitting the most important features first. Furthermore, it is able to increase throughput by exploiting available bandwidth when there is less inference (Fig. 8).

D. Performance in Changing Scenarios

This subsection further examines PNC’s ability to adapt to bandwidth variations by switching among different network scenarios dynamically during each experiment. Fig. 9 shows a consecutive sequence of 60 offloaded images with an image arrival period of $T = 500$ ms and decreasing network band-

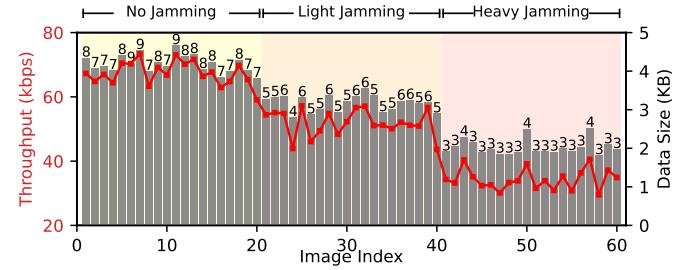


Figure 9: A typical run of the offloading process under varying network bandwidth. The red curve shows the throughput of offloading traffic. The gray bars represent the data size of the fully offloaded features. The black number above each bar represents the number of features fully offloaded to the edge server. Note that Huffman encoding encodes each feature into unequal sizes so that the number of features and data size are not always proportional to each other.

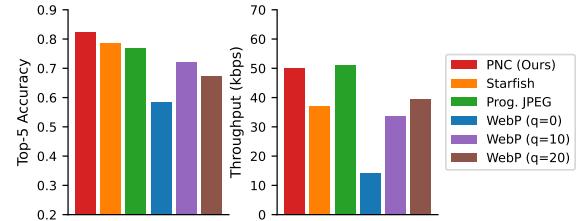


Figure 10: Image classification accuracy and offloading throughput under varying scenarios.

width. We use `iperf3` to create varying levels of interference according to the No, Light, and Heavy Jamming network scenarios in sequence, each lasting for 10 seconds. As the level of interference increases, the throughput of PNC drops along with the data size and the number of features offloaded to the server, which illustrates PNC adapting to the variations in network bandwidth by delivering a different number of features for classification.

We then conduct a set of experiments under varying bandwidth, where we switch between “No Jamming” and “Heavy Jamming” every 10s to create significant changes to available bandwidth. The performance of PNC and the baselines are shown in Fig. 10. Consistent with Fig. 6 and the corresponding analysis, PNC achieves the best classification performance, followed by Starfish and Progressive JPEG. This confirms that PNC can maintain high levels of classification accuracy by prioritizing important features in the presence of low bandwidth. Like PNC, Progressive JPEG also fully utilizes bandwidth but its encoding is optimized for reconstruction quality instead of classification accuracy. Starfish underutilizes the network and has a lower throughput, especially when no network jamming exists, but still outperforms Progressive JPEG in this experiment due to its advantage in neural encoding. The WebP baselines again perform the worst because of the impact of partially received images in the presence of heavy network jamming.

In summary, the experiments demonstrate that PNC (1) runs efficiently on resource-constrained IoT or embedded devices, (2) achieves superior classification accuracy in comparison to both standard compression methods and state-of-the-art neural compression approaches under bandwidth and deadline constraints, and (3) can adapt to bandwidth variations while maintaining high levels of classification accuracy.

VIII. CONCLUSION

In this paper, we proposed a progressive neural compression framework, PNC, for offloading images under timing constraints for edge-assisted classification. In contrast to neural compression solutions that usually encode inputs into fixed sizes, PNC is inherently designed to support progressive encoding optimized for image classification. As a result, PNC can classify images at high accuracy even when the encoded image is partially received, thereby allowing the system to adapt to different timing constraints and unpredictable bandwidth fluctuations often experienced on wireless sensor networks. Empirical evaluation of PNC on a testbed demonstrates its superior classification performance over state-of-the-art compression solutions under varying wireless conditions and timing constraints, as well as its high encoding efficiency on IoT devices. In the future, the idea of progressive neural compression may be extended to other real-time applications, *i.e.*, automatic speech recognition and object detection, that involve compression with DNNs.

ACKNOWLEDGMENT

This research was supported by the Fullgraf Foundation and in part by NSF Grant CNS-2006530.

REFERENCES

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [5] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.
- [6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.
- [7] 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.
- [8] A. J. Hussain, A. Al-Fayadh, and N. Radi, “Image compression techniques: A survey in lossless and lossy algorithms,” *Neurocomputing*, vol. 300, pp. 44–69, 2018.
- [9] A. E. Eshratifar, A. Esmaili, and M. Pedram, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.
- [10] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, “Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency,” in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.
- [11] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, “Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems,” *IEEE Access*, vol. 8, pp. 212 177–212 193, 2020.
- [12] J. Shao and J. Zhang, “Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems,” in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2020, pp. 1–6.
- [13] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, “Supervised compression for resource-constrained edge computing systems,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 2685–2695.
- [14] A. Kulkarni, A. Seetharam, A. Ramesh, and J. Herath, “Deepchannel: Wireless channel quality prediction using deep learning,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 443–456, January 2020.
- [15] P. J. Mateo, C. Fiandrino, and J. Widmer, “Analysis of tcp performance in 5g mm-wave mobile networks,” in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.
- [16] T. Koike-Akino and Y. Wang, “Stochastic bottleneck: Rateless auto-encoder for flexible dimensionality reduction,” in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 2735–2740.
- [17] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, “Temporal properties of low power wireless links: Modeling and implications on multi-hop routing,” in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 414–425. [Online]. Available: <https://doi.org/10.1145/1062689.1062741>
- [18] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, “An empirical study of low-power wireless,” *ACM Trans. Sen. Netw.*, vol. 6, no. 2, mar 2010. [Online]. Available: <https://doi.org/10.1145/1689239.1689246>
- [19] J. Zhao and R. Govindan, “Understanding packet delivery performance in dense wireless sensor networks,” in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 1–13. [Online]. Available: <https://doi.org/10.1145/958491.958493>
- [20] G. K. Wallace, “The jpeg still picture compression standard,” *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.
- [21] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.
- [22] Google, “WebP compression technique,” <https://developers.google.com/speed/webp>, 2010, accessed: 2022-03-15.
- [23] L. Theis, W. Shi, A. Cunningham, and F. Huszár, “Lossy image compression with compressive autoencoders,” *arXiv preprint arXiv:1703.00395*, 2017.
- [24] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational image compression with a scale hyperprior,” *arXiv preprint arXiv:1802.01436*, 2018.
- [25] D. Minnen, J. Ballé, and G. D. Toderici, “Joint autoregressive and hierarchical priors for learned image compression,” *Advances in neural information processing systems*, vol. 31, 2018.
- [26] Y. Lu, Y. Zhu, Y. Yang, A. Said, and T. S. Cohen, “Progressive neural image compression with nested quantization and latent ordering,” in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 539–543.
- [27] G. Toderici, S. M. O’Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, “Variable rate image compression with recurrent neural networks,” *arXiv preprint arXiv:1511.06085*, 2015.
- [28] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell, “Full resolution image compression with recurrent neural networks,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5306–5314.
- [29] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurosurgeon: Collaborative intelligence between the cloud

and mobile edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[30] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, “Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services,” *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[31] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, “Distilled split deep neural networks for edge-assisted real-time systems,” in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019, pp. 21–26.

[32] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy, “Deep variational information bottleneck,” *arXiv preprint arXiv:1612.00410*, 2016.

[33] S. Singh, S. Abu-El-Haija, N. Johnston, J. Ballé, A. Shrivastava, and G. Toderici, “End-to-end learning of compressible features,” in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 3349–3353.

[34] P. Hu, J. Im, Z. Asgar, and S. Katti, *Starfish: Resilient Image Compression for AIoT Cameras*. New York, NY, USA: Association for Computing Machinery, 2020, p. 395–408. [Online]. Available: <https://doi.org/10.1145/3384419.3430769>

[35] G. Hinton, O. Vinyals, J. Dean *et al.*, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

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

[37] A. Polino, R. Pascanu, and D. Alistarh, “Model compression via distillation and quantization,” *arXiv preprint arXiv:1802.05668*, 2018.

[38] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, “Adaptive quantization for deep neural network,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[39] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[40] A. Chakrabarti, R. Guérin, C. Lu, and J. Liu, “Real-time edge classification: Optimal offloading under token bucket constraints,” in *Proc. IEEE/ACM Symposium on Edge Computing (SEC)*, 2021.

[41] J. Qiu, R. Wang, A. Chakrabarti, R. Guérin, and C. Lu, “Adaptive edge offloading for image classification under rate limit,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.