LimitNet: Progressive, Content-Aware Image Offloading for Extremely Weak Devices & Networks

Ali Hojjat
Kiel University
Germany
aho@informatik.uni-kiel.de

Janek Haberer
Kiel University
Germany
jha@informatik.uni-kiel.de

Tayyaba Zainab
Kiel University
Germany
tza@informatik.uni-kiel.de

Olaf Landsiedel
Kiel University
Germany
ol@informatik.uni-kiel.de

ABSTRACT

IoT devices have limited hardware capabilities and are often deployed in remote areas. Consequently, advanced vision models surpass such devices' processing and storage capabilities, requiring offloading of such tasks to the cloud. However, remote areas often rely on LPWANs technology with *limited bandwidth, high packet loss rates*, and *extremely low duty cycles*, which makes fast offloading for time-sensitive inference challenging. Today's approaches, which are deployable on weak devices, generate a non-progressive bit stream, and therefore, their decoding quality suffers strongly when data is only partially available on the cloud at a deadline due to limited bandwidth or packet losses.

In this paper, we introduce `LimitNet`, a progressive, content-aware image compression model designed for extremely weak devices and networks. `LimitNet`'s lightweight progressive encoder prioritizes critical data during transmission based on the content of the image, which gives the cloud the opportunity to run inference even with partial data availability.

Experimental results demonstrate that `LimitNet`, on average, compared to SOTA, achieves 14.01 p.p. (percentage point) higher accuracy on ImageNet1000, 18.01 pp on CIFAR100, and 0.1 higher mAP@0.5 on COCO. Also, on average, `LimitNet` saves 61.24% bandwidth on ImageNet1000, 83.68% on CIFAR100, and 42.25% on the COCO dataset compared to SOTA, while it only has 4% more encoding time compared to JPEG (with a fixed quality) on STM32F7 (Cortex-M7).

KEYWORDS

Deep Learning, Edge Computing, Lightweight AutoEncoders, Content-Aware Encoding, Image Compression, Progressive Offloading, Progressive Compression, Internet of Things

Figure 1: Qualitative comparison of `LimitNet`'s progressive reconstruction. `LimitNet` detects the important parts of the image and sends the encoded data in the order of importance. Progressive offloading allows the cloud to run the inference at any point (lightning symbol). The horizontal axis shows the offloading bitstream of an image, the first number shows the size of received data (in KB) at a specific time, followed by the corresponding Top5 Accuracy (Top5-Acc) of EfficientNet-B0 [61] on ImageNet1000 [27] using this received data.

1 INTRODUCTION

Motivation: Neural networks enable new computer vision applications, such as classification and object detection, in the Internet of Things (IoT). For example, AI-enabled distributed IoT cameras enable emergency systems such as Apple's emergency SOS [8] disaster response [63, 105], intruder detection [96], fire detection [10], wild life monitoring [19] and road security surveillance [82]. In such scenarios, it is crucial to have the classification result within a specific **deadline** to ensure a timely response [25] such as triggering alerts.

Challenges: IoT devices are often deployed in remote areas and consist of embedded devices with **minimal power and hardware capabilities** to limit costs. As a result, modern vision models [29,

Table 1: Comparison of compression-offloading methods. "Offloading Granularity" measures the scale of the individual transmitted segments, and "Incomplete Data Accuracy" refers to the accuracy of the model when only a portion of data is available for decoding on the cloud side.

Model	Offloading Granularity	Transmission Cost	Incomplete Data Accuracy	Local Cost	Objectives
DeepCOD [103]	-	Med.	Low	Low	Accuracy
BottleNet++ [88]	-	High.	Low	Low	Accuracy
AgileNN [44]	-	Med.	Low	Low	Accuracy
SPINN [52]	Filter	Med.	Low	Low	Accuracy
FLEET [42]	Set of filters	Med.	High	Low	Accuracy
DynO [7]	-	Low	Low	High	Accuracy
Starfish [41]	-	Low	Med.	Med.	Accuracy, Perception
JPEG [98]	-	Low	Low	Low	Perception
ProgJPEG [98]	DCT Scan	Low	Med.	Low	Perception
Ballé *et al.* [13]	-	Low	Low	High	Perception
Full-Res [94]	Latent	Low	Med.	High	Perception
DCCOI [22]	-	Med.	Low	Low	Accuracy
AccelIR [104]	-	Low	Low	High	Accuracy
`LimitNet`	**Subfilter**	**Low**	**High**	**Low**	**Accuracy**

36, 83] quickly exceed embedded systems' computing capabilities. Therefore, the input image must be compressed first [41, 88, 97, 103], and then sent to the cloud for further processing. However, remote areas commonly have limited internet access, i.e., they often do not have access to cellular networking. In such settings, communication is usually limited to LPWAN technology [21] (e.g., Sigfox [53] and LoRa [15]) with **very low and often dynamic bandwidth** (less than 50 KB/s) as satellite communication is often too costly and energy-intensive [21, 32]. In these scenarios, IoT cameras use shared communication links, forcing them to have **limited duty cycles** (less than 1%). In addition to these limitations, LPWANs have a **high packet loss rate**, which increases the transmission time by forcing multiple retransmissions [89]. Therefore, LPWAN cannot guarantee to transmit all data to the cloud within a predefined timeframe, e.g., a deadline or a given duty cycle budget.

To overcome these challenges, the system must be able to run inference on the cloud even with *incomplete data*.

Approach: When it comes to dealing with incomplete data, standard compression-offloading methods are not a solution since they produce an *atomic latent*, which necessitates having all data for decoding. **Progressive Encoding**, known as *fine-grained scalability (FGS)* [62, 86], is the common solution that enables a stepwise transmission of data, starting from low-resolution or coarse-grained information and gradually improving the quality or level of detail during transmission. Progressive encoding helps to prioritize critical data, optimize bandwidth usage, and, as a result, ensures that partial data, i.e., if the transmitter cannot transmit the complete compressed image before the deadline, can be efficiently used on the cloud side. Progressive compression [11, 13, 38, 77, 94], however, imposes a significant computational load due to its extensive parameterization, often comprising millions of parameters. To address this, creating a lightweight encoder – as done in this work – becomes essential, especially for weaker devices with hardware limitations.

`LimitNet`: In this paper, we introduce `LimitNet`, a lightweight, progressive, and content-aware image compression-offloading model designed for weak devices like Cortex-M33 or M7 series MCUs under extremely limited networks such as LPWANs. In `LimitNet`, we first design a lightweight CNN to encode the input into a latent representation. Next, by designing a lightweight saliency detector network, we identify the important parts of the input image. However, using the saliency map out of the box leads to poor accuracy. To effectively use the saliency map, we introduce a "Gradual Scoring" algorithm, which enables the model to *learn and choose* how much background, i.e., context, it needs during the training for better classification results; see Fig. 1. The output of Gradual Scoring is an importance score for each latent data point, indicating their relative impact on classification results. Afterward, in order of the scores, we progressively transmit the data to the cloud. At any given time, the cloud can reconstruct the image and run the inference, which is essential when we have a deadline and deal with limited bandwidth. Also, in the case of packet loss, `LimitNet`'s prioritized approach ensures that the cloud side receives the most crucial parts for accurate classification.

As highlighted in Table 1, in comparison to the state of the art, `LimitNet` is the only method characterized by both low transmission and low local computational costs. Its progressiveness operates at a granularity level as small as a subfilter, achieving notably high accuracy even when only a portion of the encoded data is accessible on the cloud, as we show in our evaluation in Section 4.

Contributions:

(1) `LimitNet` is a *progressive content-aware encoder* incorporating a lightweight saliency detector and a novel Gradual Scoring mechanism. It is highly suitable for LPWANs, given

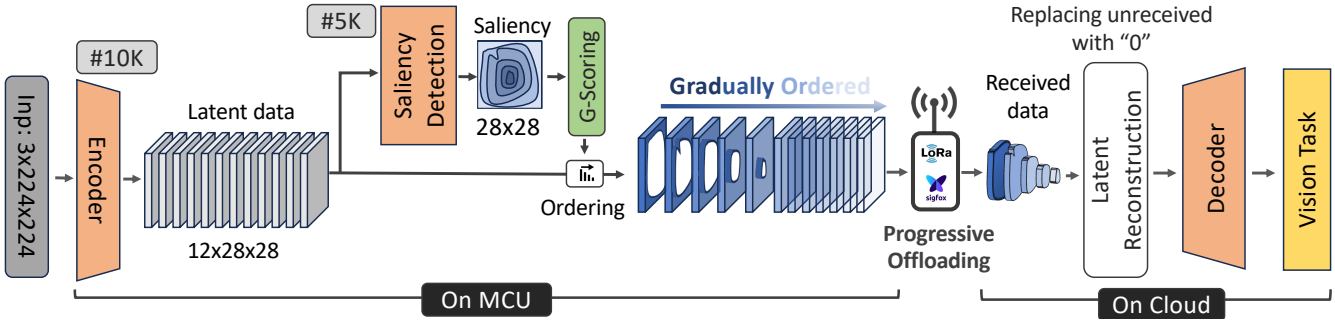


Figure 2: Overview of LimitNet. Our encoder compresses the input to a latent representation. Gradual Scoring and our saliency detector extract the important parts of the latent data and assign an importance score to each latent data point. Afterward, LimitNet starts to transmit the latent to the cloud in order of its importance score. On the cloud side, at any given time, we can reconstruct the latent by setting unreceived values to zero and feeding it to a powerful decoder. After reconstruction, we feed the output to a vision model [61, 83].

their limited and dynamic network bandwidth, restricted duty cycles, and high packet loss rate (Sections 3.3, 3.4, 4.2 and 4.4).

(2) LimitNet is a *very lightweight image encoder* with 15K parameters, efficiently executable on extremely weak devices, such as ARM Cortex-M series (Section 3.2).

(3) We evaluate the performance of LimitNet on ARM Cortex-M33 and M7 for vision tasks such as classification and object detection and evaluate system-level metrics such as RAM, Flash, current usage, execution time, and power consumption. We also evaluate LimitNet's performance under different network bandwidths and different packet loss rates by comparing it with SOTA models (Sections 4.2, 4.3 and 4.4).

Summary of results:

(1) *Accuracy/mAP results*: For a given data size, LimitNet on average achieves up to 14.01 p.p. (percentage point) higher accuracy compared to the SOTA on the ImageNet1000 dataset, 18.01 p.p. accuracy improvement on the CIFAR100 dataset, and 0.1 mAP@0.5 improvement on the COCO dataset.

(2) *Rate results*: For a given accuracy or mAP@50, LimitNet on average saves 61.24% bandwidth compared to the SOTA on the ImageNet1000 dataset, 83.68% on the CIFAR100 dataset, and 42.25% on the COCO dataset.

(3) *System benchmarks*: We deploy our model on two microcontrollers, nRF5340 (Cortex-M33) and STM32F7 (Cortex-M7). Our results show that LimitNet needs 260 ms, 360 KB (17%) RAM, and 107 KB (5%) Flash to run on STM32F7 and 2189 ms, 344 KB (33%) RAM, and 102 KB (10%) Flash to run on nRF5340. LimitNet only uses 4% and 11% more encoding time compared to JPEG (with a fixed quality) on STM and nRF, respectively, while, unlike JPEG, producing a progressive bitstream.

2 BACKGROUND AND MOTIVATION

This section presents the required background, limitations, and challenges of existing compression models and LPWANs.

2.1 Compression models

2.1.1 Stand-alone image compression models. We categorize image compression models into two groups: classical approaches such as JPEG [98], JPEG2000 [90], WebP[102], BPG [16], and more recent ones which use the power of deep learning for compression, including Hyperprior [13], Full-Resolution [94] and other methods [12, 23, 77, 91, 104]. While recent deep compression models exhibit excellent performance, they cannot be utilized on embedded MCUs due to their stark resource demands. Starfish [41] addresses these constraints by introducing a lightweight image encoder. Further, it incorporates dropout in the bottleneck to account for the network's unreliability. To the best of our knowledge, Starfish is the only deep image compression approach that is executable on the MCUs and addresses the dynamic nature of wireless networks. In addition to having a light encoder, we argue that it is crucial to prioritize important data when it comes to limited bandwidths, which Starfish cannot do since it has a content-agnostic encoder. In contrast, LimitNet features a lightweight, progressive, content-aware encoder, which prioritizes the important data during the offloading. As a result, it ensures that data arrives in the order of importance, allowing the cloud to gracefully run inference even with partial data availability.

2.1.2 Compression for offloading. In resource-constrained networks, offloading data to a more computationally capable node is a potential solution to overcome resource challenges on these devices. Besides image compression models, numerous works focus on data compression in offloading settings, i.e., compressing the data before offloading it to the cloud for further processing. In general, we can group the offloading techniques into two categories [74]: offloading with autoencoders [24, 31, 40, 45, 70, 71] and without

autoencoder [30, 47, 58, 59, 80, 106]. The autoencoder-based models such as DeepCOD [103] and BottleNet++ [88] are more suitable for the situation where we have limited network bandwidth since they compress the data before offloading.

Despite their advantages, these offloading models produce a non-progressive content agnostic bitstream, which needs all data for decoding and does not consider the effect of each data point in the target task. In contrast, `LimitNet` features a progressive, content-aware encoder, which prioritizes the important data during the offloading. It ensures that data arrives in the order of importance, allowing the cloud to gracefully run inference even with partial data availability

2.2 LPWAN

LPWANs, including Sigfox, LoRaWAN, and NB-IoT, are wireless communication technologies specifically designed to facilitate long-range, low-power connectivity in the IoT field. In contrast to cellular technologies and WiFi, LPWANs utilize lower frequencies to efficiently transmit data over extended distances: Sigfox covers up to 40km, LoRaWAN up to 20km, and NB-IoT up to 10km. However, LPWANs provide very limited bandwidth: Sigfox offers 100 B/s with a 1% duty cycle, LoRaWAN provides 0.3 KB/s to 50 KB/s also with a 1% duty cycle, and NB-IoT delivers up to 200 KB/s [15, 21, 53]. For example, sending a 10 KB image using Sigfox takes roughly 800 seconds at 100 B/s. Similarly, with LoRaWAN, at 10 KB/s, the process takes about 8 seconds. However, due to the 1% duty cycle, a node can only use the network for 36 seconds per hour on average, which becomes quickly noticeable when, for example, dealing with multiple image transmissions in time-sensitive scenarios. Further, due to wireless link dynamics, connection quality might unexpectedly deteriorate, forcing the LPWAN to retransmit data and potentially even change to a more robust encoding, which both, in turn, provide less bandwidth and thereby increase transmission times and the radio duty cycle [15, 89]. Therefore, LPWAN cannot guarantee to transmit all data within a timeframe, e.g., a deadline or a given duty cycle budget. Thus, it is essential to be able to operate on partial data.

`LimitNet` addresses these issues by incorporating a *progressive encoder* that identifies important parts of the latent space for classification accuracy. By sending data in order of importance and retransmitting it as needed until the time budget for transmission, i.e., deadline or duty cycle, is reached, `LimitNet` ensures that the most important data is available at the cloud, and it gracefully produces an output image from partial data.

3 LIMITNET

We begin this section by presenting the key components of `LimitNet` before detailing its architecture, training phases, offloading mechanism, and quantization process.

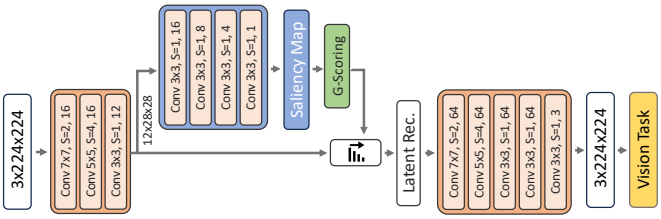


Figure 3: The architecture of `LimitNet` is inspired by ResNet [36] and adopts larger kernels for the first layers. All the used layers are supported by the DSP accelerator of the MCUs.

3.1 Overview

In `LimitNet`, we first encode the input image to a latent space utilizing a lightweight encoder; see Fig. 2. Then, with a lightweight saliency detection branch, we detect the important parts of the input image. However, using this saliency map directly on the latent representation causes bias to the foreground, resulting in blocky outputs and subpar accuracy, see Fig. 5. To address this, we introduce a "Gradual Scoring" algorithm, which enables the model to learn and determine the necessary background dynamically. The output of Gradual Scoring is an importance score for each data point in the latent, indicating their relative impact on the classification accuracy. Afterward, based on these scores, we progressively transmit the data to the cloud, where we run the image decoder and the classifier. The cloud can reconstruct the image and gracefully run inference at any time. In the case of packet loss, `LimitNet`'s prioritized approach ensures that the cloud side receives the most important data first.

3.2 Lightweight Encoder

We design an asymmetric autoencoder [50, 103] with a lightweight encoder $ENC_{\theta_{ENC}}$ for the edge and a deeper decoder $DEC_{\theta_{DEC}}$ for the cloud. The encoder gets the input image $X^{C\times H\times W}$ and compresses it to $Z^{L\times K\times K}$:

$$X^{C\times H\times W} \rightarrow ENC_{\theta_{ENC}}(X^{C\times H\times W}) \rightarrow Z^{L\times K\times K} \tag{1}$$

Where $C \times H \times W$ shows the input size, and $L \times K \times K$ shows the shape of the latent data. Inspired by ResNet [36], we use large kernels (3, 5, and 7) for the first layers; see Fig. 3.

3.3 Saliency Detection

After data encoding, we need to detect the important parts of the input image to add priority to their corresponding encoded data.

There are several ways to detect the important parts, such as ROI detection, Explainable-AI, and Saliency Detection, see Section 5. However, running these models on embedded devices is often infeasible due to their high computational and resource costs, comparable to or even greater than running a full classifier. Despite the hardware limitations, we leverage the benefits of edge computing by employing a lightweight model that *mimics* the output of

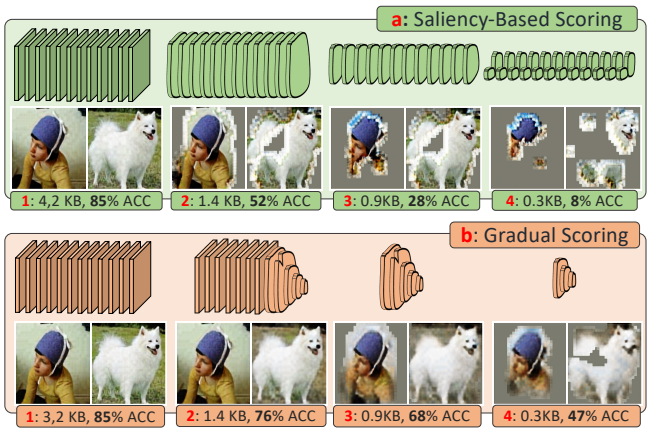


Figure 4: Reconstruction outputs and Top5-Acc (on ImageNet1000) at different data availability levels when we only use the saliency map compared to combining it with Gradual Scoring.

a complex saliency detector. We design a lightweight, 4-layer network $\text{SalDet}_{\theta_{SalDet}}$ for saliency detection and train it via *Knowledge Distillation (KD)* [37] using BASNet [81], one of the state-of-the-art saliency detection models, as teacher model. This branch takes latent $Z^{L\times K\times K}$ and extracts the saliency map $I^{K\times K}$, see Eq. 2.

$$Z^{L\times K\times K} \rightarrow \text{SalDet}_{\theta_{\text{SalDet}}}(Z^{L\times K\times K}) \rightarrow I^{K\times K} \tag{2}$$

Our saliency detection branch, (θ_{SalDet}), has only 5k parameters, which is 0.001% of the parameters of the original saliency model. Although our lightweight model does not achieve the same level of performance as BASNet, the resulting importance map provides essential information about the important regions, which greatly enhances the model's accuracy, as our evaluation in Section 4.3 shows.

3.4 Gradual Scoring: The Devil is NOT in the Details!

After extracting the saliency map, a naïve solution is to use this map to score each encoded data point. However, relying only on the saliency map leads to fragmented and blocky outputs and hence, poor accuracy. For instance, examples *a*2, *a*3, and *a*4 in Fig. 4 illustrate the decoder outputs with varying levels of available data. As shown in these examples, fragmentation occurs since we only transfer the parts with the highest score in the saliency map, focusing only on the foreground and sending the same locations across the latent for all filters. Consequently, the decoder can reconstruct these parts only, which hinders the classification model's ability to accurately identify objects due to the *absence of contextual information* [17]. To effectively use saliency, we introduce a Gradual Scoring algorithm, which enables the model to *learn and choose* how much background, i.e., context, it needs during the training for better classification results. Our results show that incorporating Gradual Scoring leads to significantly improved classification accuracy. For instance, examples *a*3 and *b*3 in Fig. 4 show that for the same data size, gradual scoring leads to 40 p.p. higher accuracy on the ImageNet1000 [27].

We design the Gradual Scoring mechanism inspired by the gradual ordering in TailDrop [49], which uses dropout to order the information in the latent. The Gradual Scoring function takes the saliency map $I^{K\times K}$ and by adding a *descending constant value*, G_{Factor}, produces a $S^{L\times K\times K}$ tensor, which shows the importance score of each data point in the latent; see Eq. 3.

$$I^{K\times K} \rightarrow \text{GS}(I^{K\times K}) \rightarrow S^{L\times K\times K} \tag{3}$$

As illustrated in Fig. 5, the score of each latent data point is calculated as follows:

$$S_{i,[0:K],[0:K]} = I_{[0:K],[0:K]} + G_{Factor} \times i, \quad \forall i \in \{0, 1, 2, \ldots, L\} \tag{4}$$

To get the best results, we need to employ Gradual Scoring in the training pipeline. To do so, in each training step, we randomly select a value $p = \mathcal{U}(0, 100)$, and zero out the p% of the latent $Z^{L\times K\times K}$ with the *lowest score* in the score tensor $S^{L\times K\times K}$; see Eq. 5 and Eq. 6.

$$Z^{L\times K\times K}, S^{L\times K\times K} \xrightarrow{\text{Dropping}} Z'^{L\times K\times K} \tag{5}$$

$$Z'_{i,j,k} = \begin{cases} Z_{i,j,k} & \text{if } S_{i,j,k} \geq p^{th}\text{largest value of } S^{L\times K\times K} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$Z'^{L\times K\times K}$ represents the modified version of $Z^{L\times K\times K}$ that contains the p% of the highest important scores.

3.5 Offloading, Decoder and Classifier

After encoding, we quantize the latent data with 6 bits. We also downsize the saliency map to an 8×8 representation and then quantize it with 5 bits, as we do not lose a lot of accuracy and add considerable compression. We initiate the offloading process by first transmitting the saliency map before sending the encoded data. The overhead of transmitting this map is a maximum of 40 bytes, which is practically negligible. For instance, in an LPWAN network with a throughput of 5 KB/s, it takes less than 1 ms to transmit. This step is crucial as the decoder requires precise placement information for reconstructing each data point. On the cloud side, after receiving the compressed image data, we fill any unreceived latent values with zero based on the saliency map and reconstruct the latent $\hat{Z}^{L\times K\times K}$. While we prioritize minimizing computational costs during encoding, we are less concerned about the costs of decoding. We employ a decoder $DEC_{\theta_{DEC}}$ comprising a 5-layer convolution with different kernel sizes (7, 5, and 3). These varying kernel sizes allow the decoder to capture features at different scales and reconstruct the finer details of the original image, see Fig 3.

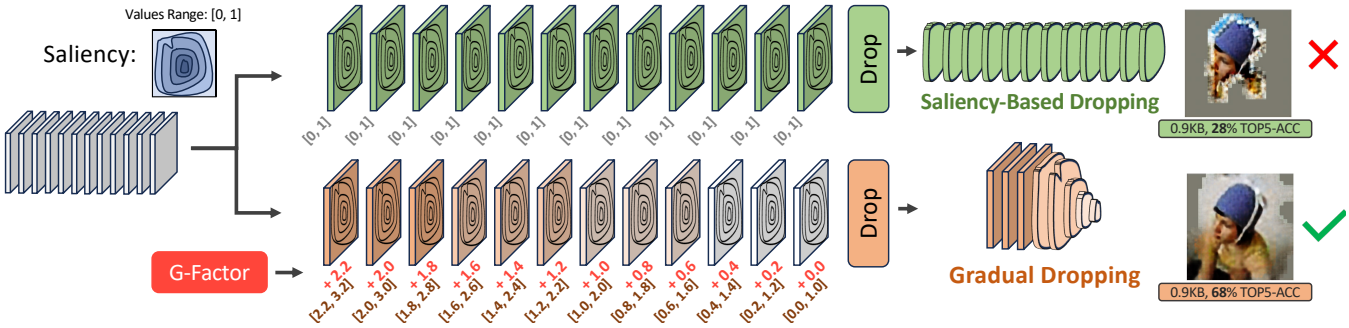


Figure 5: Details of Gradual Scoring. This Figure illustrates where and how we add G_{Factor} to each filter's activations of the latent, enabling the model to learn and choose how much background, i.e., context, it needs.

Table 2: Training phases of `LimitNet`

Phase	Epochs	LR	Loss Function	Traning Notes
1	100	0.001	Rec Loss + Saliency Loss	KD for the saliency detection, training with Gradual Scoring
2	6	0.00005	CLS Loss	Training with Gradual Scoring, freezing the CLS

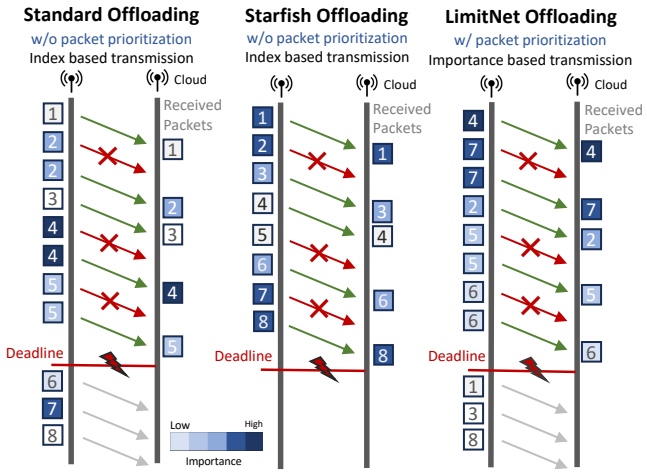


Figure 6: A high-level illustration of `LimitNet`'s transmission-retransmission policies. In conventional offloading schemes, we transmit the data order of its index, and in the case of packet loss, either we skip (like Starfish) or retransmit data with the same policy (standard offloading). `LimitNet`, in contrast, transmits and retransmit packets based on their importance. Thereby, it ensures that the most important packets arrive first, essential for graceful progressive decoding.

This decoder is responsible for reconstructing the original image $X^{C\times H\times W}$. Once the image is reconstructed, we feed it to the vision model $DEC_{\theta_{DEC}}$ and predict the label $\hat{y}$; see Eq. 7 and Eq. 8. Specifically, in this paper, we utilize EfficientNet-B0 [92] and YOLOv5 [83], depending on the evaluation scenario.

$$Z'^{L\times K\times K} \xrightarrow{\text{Quant., Huffman Enc.}} \hat{Z}'^{L\times K\times K} \tag{7}$$

$$\hat{Z}'^{L\times K\times K} \xrightarrow{\text{Lat. Rec.}} \hat{Z}^{L\times K\times K} \xrightarrow{DEC_{\theta_{DEC}}} \hat{X}^{C\times H\times W} \xrightarrow{CLS_{\theta_{CLS}}} \hat{y} \tag{8}$$

3.5.1 Prioritized Packet Transmission-Retransmission. In `LimitNet`, we offload packets based on their importance score produced by Gradual Scoring. However, as discussed in Section 2.2, LPWAN links are highly dynamic and often have a high packet loss rate, necessitating retransmissions.

Due to its content-aware progressive bitstream, the importance of each packet in `LimitNet` is known. Therefore, we transmit and – if needed – retransmit packets in order of importance until an application-specific deadline is reached or our duty cycle budget is over. As a result, this ensures that the receiver always receives the most important data in the case that we cannot transmit the entire latent. Looking at Fig. 6, through `LimitNet`'s prioritized offloading, critical packets (4, 7, 2, 5, and 6) are delivered to the cloud before the deadline. In contrast, standard offloading schemes send packets based on their index order without considering their importance, resulting in the cloud missing crucial packets (e.g., packets 6 and 7). Although Starfish has regional importance for mitigating packet loss, it also lacks packet prioritization capabilities. Thus, similar to the conventional offloading models, it transmits packets in random order and potentially loses crucial packets (e.g., packets 2 and 7).

3.6 Application Scenario

Distributed IoT cameras, such as alarm systems, utilize a shared LPWAN connection for transmitting data to the cloud, necessitating limited duty cycles. In conventional approaches, inference needs to wait until all data is received, where it takes approximately 100 seconds to send a 5KB image through a 5 KB/s LoRa link with a 1% duty cycle. However, in `LimitNet`, each node compresses images progressively, sorts encoded data by importance, and transmits

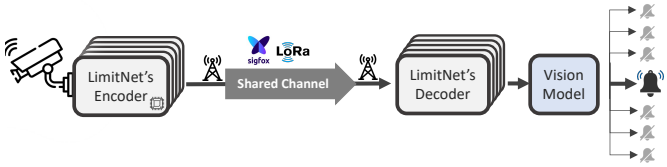


Figure 7: A high-level application scenario for LimitNet. In this scenario, multiple cameras transmit their images through a shared LPWAN link.

them accordingly within its duty cycle. On the cloud end, after each cycle, the decoder can reconstruct the image, conduct inference for each camera, and activate the alarm system if needed, without needing to wait for the complete data to be received; see Fig. 7.

4 EVALUATION

In this section, we evaluate the performance of LimitNet and compare it to state-of-the-art image compression and offloading models. We focus on vision tasks like classification and object detection, use accuracy and mAP as evaluation metrics, and assess system-level benchmarks on MCUs as follows:

Accuracy vs. Data Size: We evaluate LimitNet's accuracy on different compression ratios and data sizes and compare it to SOTA models.

Saliency Detection Evaluation: We evaluate the saliency branch and the Gradual Scoring algorithm to analyze how well they identify important parts.

System-level Benchmarks on MCU: We implement LimitNet on STM32F7 (Cortex-M7) and nRF5340 (Cortex-M33) and evaluate system-level benchmarks. In our experiments, we focus on Flash usage, RAM usage, execution time, and energy consumption. Furthermore, we evaluate LimitNet and SOTA models' performance under LoRaWAN networks.

MS-SSIM and PSNR: While it is common to use visual metrics such as MS-SSIM and PSNR when comparing image compression models, LimitNet only produces a partial reconstruction (see Fig. 1). As these metrics measure the average difference of all pixels, the result will be significantly biased to the missing values. However, the main objective of LimitNet is classification accuracy and not the perceptual quality. Therefore, we do not evaluate these metrics.

Note that, for simplicity, we evaluate the performance of LimitNet under the assumption of a single image in the transmission queue, as is common in the literature [41–44]. However, LimitNet can also be extended to handle multiple images, either in parallel or sequentially, which we leave for future work.

4.1 Experimental Setup

4.1.1 Implementation. We implement LimitNet in TFLite-Micro [2] and Zephyr RTOS [3] for STM32F7 [6] and nRF5340 [5] MCUs. The STM has 2 MB of Flash, 1 MB of RAM, and a dual-core configuration consisting of an ARM Cortex-M7 and an ARM Cortex-M33, which can be clocked up to 480 MHz and 240 MHz, respectively. We deploy LimitNet on the Cortex-M7. The nRF has 1 MB Flash, 512 kB RAM, and a dual ARM Cortex-M33 core that can be clocked at 128 and 64 MHz. Both MCUs provide accelerated integer inference via DSPs and CMSIS-NN. To effectively utilize LimitNet and ensure that the model's weights do not excessively consume memory resources, we apply post-training integer quantization to LimitNet's image encoder, i.e., we quantize the weights to 8-bit integer representations. LimitNet is available as open source[1].

4.1.2 Datasets. We evaluate LimitNet on three public datasets: To show the maximum capacity of our model, we train and evaluate LimitNet on ImageNet1000 [27]. We also fine-tune and evaluate our model on CIFAR100 [51]. To investigate the performance of LimitNet on harder tasks such as object detection, we also evaluate the pre-trained ImageNet1000 model on the COCO dataset [66] by using YOLOv5 [83]. For all datasets, we resize the input images to 224×224.

4.1.3 Training and hyperparameters. Our training process has two phases. At first, we train the autoencoder and the saliency branch jointly using Gradual Scoring with $G_{Factor} = 0.2$.

In the second phase, we stitch the classification model after the decoder and train the whole network on classification loss, see Table 2.

4.2 Compression Efficiency

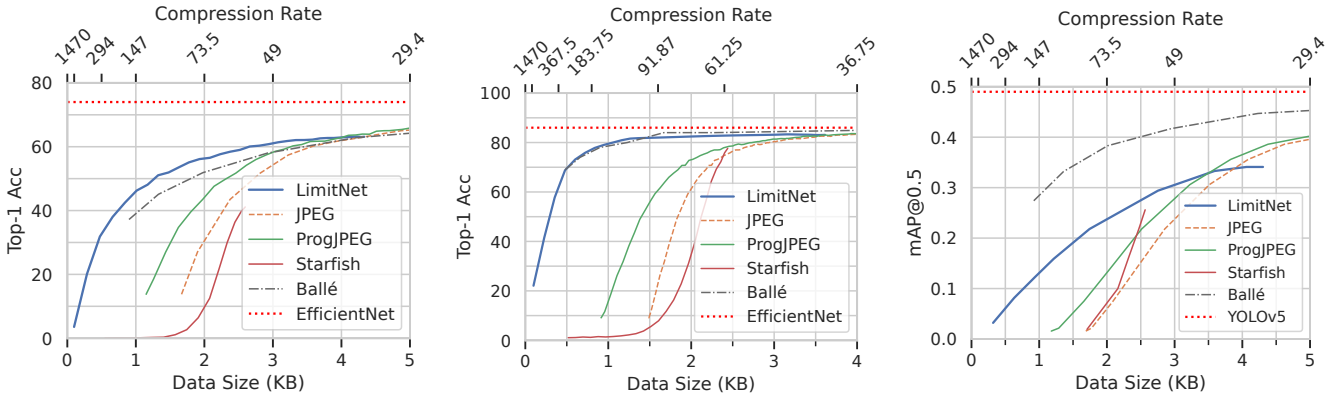
Baselines: In this section, we compare the compression efficiency of LimitNet to three sets of baselines: (1) resource-efficient compression models deployable on embedded MCUs: Starfish [41] as the only existing image compression model that can handle partial data availability, JPEG [98] as the standard baseline, and the progressive version of JPEG (referred as ProgJPEG) as the progressive standard baseline. (2) SOTA image compression models: Ballé *et al.* [13] as the non-progressive image compression which has 165 times more parameters and 50 times more GFLOPs compared to LimitNet, see Table 3. Although there are newer and improved models, we choose Ballé as it is easy to evaluate, well-documented, and has comparable performance to SOTA. (3) Non-progressive offloading models: BottleNet++ [88] as the standard benchmark and DeepCOD [103] as SOTA.

We found that post-training integer weight quantization has less than 0.01 p.p. effect on the results, which is negligible. Given this, we only present the non-quantized results.

Metrics: In our evaluation, we use these metrics for comparison:

4.2.1 Data size vs. Accuracy/mAP. We evaluate LimitNet and other benchmarks by feeding their decoded image into EfficientNet-B0 [61] as a backend model for classification. Fig. 8a and Fig. 8b plot Top-1 classification accuracy and show that LimitNet consistently outperforms the baseline models. For example, on CIFAR100, with 1 KB of data, LimitNet achieves 80% accuracy while ProgJPEG

[1] https://github.com/ds-kiel/LimitNet



(a) Data Size vs. Accuracy on ImageNet (b) Data Size vs. Accuracy on CIFAR (c) Data Size vs. mAP@0.5 on COCO

Figure 8: Performance evaluation of `LimitNet` compared to JPEG, ProgJPEG and Starfish on ImageNet1000, CIFAR100 and COCO. We also compare it to the SOTA image compression model, Ballé *et al.*, which has 165 times more parameters than `LimitNet` and is not executable on MCUs.

Table 3: Comparing GFLOPs and the #parameters of `LimitNet` with Ballé *et al.*, the state-of-the-art image compression model, and Starfish, a MCU-based deep image compression model.

Model	#GFLOPs	#Params
`LimitNet`	**0.004**	**15K**
Starfish	0.77	78*K*
Ballé *et al.*	1.95	2.5*M*

achieves 16% accuracy. At high compression rates, `LimitNet` even outperforms Ballé *et al.* on ImageNet1000 and achieves an on-par performance on CIFAR100. We also evaluate `LimitNet` on object detection using YOLOv5 [83] as the backend model on the COCO dataset. As shown in Fig. 8c, `LimitNet` outperforms the baselines at high compression rates when compressed image sizes are 3.5 KB and smaller. This superior performance primarily stems from `LimitNet`'s ability to reconstruct the image components crucial for classification. In contrast, other progressive models like Starfish and ProgJPEG are content-agnostic and transmit images without prioritizing the crucial regions regarding classification.

Additionally, we evaluate `LimitNet` by comparing it with non-progressive offloading models such as DeepCOD [103] and BottleNet++ [88] when 100% of the data is available for decoding. We choose these models' first offloading point, ensuring implementation feasibility on MCUs. As shown in Fig. 9, `LimitNet` achieves comparable performance to these non-progressive models, despite being a progressive encoder[2].

4.2.2 BD-Rate/BD-Acc/BD-mAP. To summarize the results on compression efficiency, we utilize the Bjontegaard Delta (BD) [14], a metric for comparing various model performances over different data sizes:

[2]As common, when comparing to DeepCOD, we plot Top5-Acc.

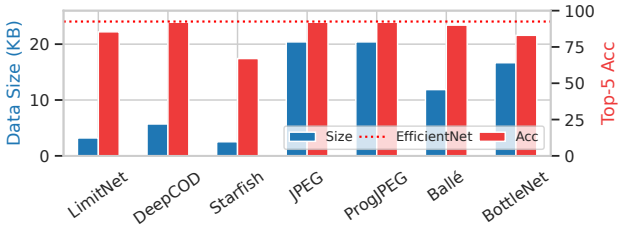


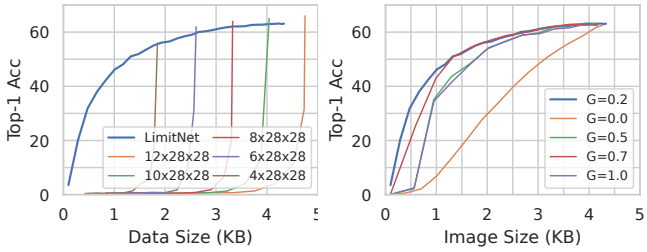
Figure 9: `LimitNet` performance evaluation, compared to non-progressive offloading models (DeepCOD [103], BottleNet++ [88], and Ballé *et al.* [13]) and other benchmarks [41, 97], when 100% of the encoded data is available. `LimitNet` achieves comparable performance to these non-progressive models despite being a progressive encoder.

- **BD-Rate** shows, on average, how much *bandwidth* a baseline saves over another baseline for a given *quality* in percent.
- **BD-Acc** shows, on average, how much a baseline improves the *accuracy* in percentage points (p.p.) over another baseline for a given *data size*.
- **BD-mAP** shows, on average, how much *mAP* a baseline improves over another baseline for a given *data size*.

Table 4 presents the BD metrics for `LimitNet` compared to JPEG, ProgJPEG, and Starfish. *Accuracy/mAP results*: for a given data size, `LimitNet` on average achieves up to 14.01 percentage points higher accuracy compared to the SOTA on the ImageNet1000 dataset, 18.01 percentage points accuracy improvement on the CIFAR100 dataset, and 0.1 mAP@0.5 improvement on the COCO dataset. (2) *Rate results*: For a given accuracy or mAP@50, `LimitNet` on average saves 61.24% bandwidth compared to the SOTA on ImageNet1000 dataset, 83.68% on the CIFAR100 dataset, and 42.25% on the COCO dataset.

Table 4: Comparison of LimitNet with baselines in terms of BD-Rate (in percent), BD-Acc (in percentage points), and BD-mAP@50. Each number in each column shows the improvement of LimitNet compared to the corresponding model.

Model	ImageNet1000		CIFAR100		COCO	
	BD-Rate (%)	BD-Acc (p.p.)	BD-Rate (%)	BD-Acc (p.p.)	BD-Rate (%)	BD-mAP (mAP@50)
ProgJPEG	61.24	14.01	83.68	18.01	42.45	0.1
JPEG	69.02	15.41	87.7	15.21	53.64	0.12
Starfish	85.09	60.33	89.81	72.26	55.02	0.15



(a) (b)

Figure 10: (a) Comparing LimitNet and Fixed-Rate AEs results on incomplete data: LimitNet outperforms Fixed-Rate AEs when we do not have all the encoded data for inference. (b) Evaluating different G_{Factor}. $G_{Factor} = 0.2$ achieves the best performance, effectively balancing object details and context in the reconstructed image.

4.3 Evaluating LimitNet in Detail

In this section, we conduct a comprehensive evaluation of each of LimitNet's components to measure the benefits of content-aware encoding, assess the impact of our Gradual Scoring mechanism using different G_{Factor} values, and evaluate the performance of our saliency detection branch.

4.3.1 Advantages of Content-Aware Encoding. We begin by assessing the impact of progressive content-aware encoding on the model results. For this, we train a set of non-progressive autoencoders with different latent sizes and run the inference on incomplete data. Fig. 10a illustrates that a normal, i.e., non-progressive, autoencoder shows a significant drop in accuracy when reconstructing the image with incomplete data, while in LimitNet, the accuracy gracefully degrades due to its progressive nature and having a content-aware encoding.

4.3.2 Gradual Scoring. In this section, we evaluate the effect of Gradual Scoring on classification accuracy. We train our model with different G_{Factor} and measure its accuracy on the ImageNet1000 dataset. As Fig. 10b shows, if we set $G_{Factor} = 0$, i.e., using the saliency map naively, it leads to a poor accuracy since it produces a blocky output. On the other hand, if we set $G_{Factor} = 1.0$, the model diminishes the effect of the saliency map, which hinders it from achieving its maximum accuracy. By choosing an intermediate value for the G_{Factor}, we allow the model to balance foreground and background in the reconstructed image. Our experiments show that $G_{Factor} = 0.2$ produces the best results; see Fig. 10b.

4.3.3 Evaluation of Saliency Detection. As discussed in Section 3.5, in the quantization step, we downsize the 32×32 maps to 8×8 for better compression and encode these in 5 bits. Therefore, when evaluating the saliency maps, we also need to evaluate the impact of this quantization. In Fig. 11, we compare the input image, saliency ground truth, and LimitNet's saliency output with and without quantization. We use BASNet [81], i.e., our teacher model, as ground truth. As this figure shows, the saliency detection branch is not as accurate as the ground truth; nevertheless, it can identify the *area* of the important parts of the image. For example, in the fourth column, although the saliency can not precisely localize the dog, it detects the parts of the image that include the target object, which bumps up the accuracy when we prioritize these sections in progressive offloading.

4.4 System-Level Benchmarks

Next, we evaluate the system-level performance of LimitNet, including current consumption, energy tracing, resource consumption, and performance under limited network bandwidths such as LoRaWAN. Although Starfish is tailored for limited hardware settings, it cannot run on the selected MCUs due to its extensive RAM usage. Therefore in this section, we only evaluate LimitNet and JPEG.

4.4.1 Resource Consumption. In Table 5, we report the resource requirements and inference time for the float32 model and the quantized int8 model with and without DSP support for NN inference acceleration on nRF5340 and STM32F7 MCUs. Only the int8 version of the encoder is deployable on MCU due to space limitations, while we can deploy the saliency detection branch with either float32 or int8 model weights. For int8, the encoder consumes 344 KB RAM (33% of RAM), about 100 KB Flash (10% of Flash), and takes 1,969 ms to run on nRF with DSP acceleration enabled and 360 KB RAM (17% of RAM), 107 KB Flash (5% of Flash), and 237 ms to run on STM with DSP acceleration enabled. Also, the saliency detection branch as quantized int8 model needs 26 KB RAM (2% of RAM), 102 KB Flash (10% of Flash), and 220 ms to run on nRF5340 with DSP acceleration enabled, and 26 KB RAM (1% of RAM), 107 KB Flash (5% of Flash), and 23 ms to run on STM with DSP acceleration enabled. Overall, LimitNet compresses images within 260 ms on the faster Cortex-M7 while it needs about 2,189 ms on the slower Cortex-M33.

4.4.2 Current Consumption and Energy Trace. To gain insights into the energy demands of LimitNet, we measure energy consumption

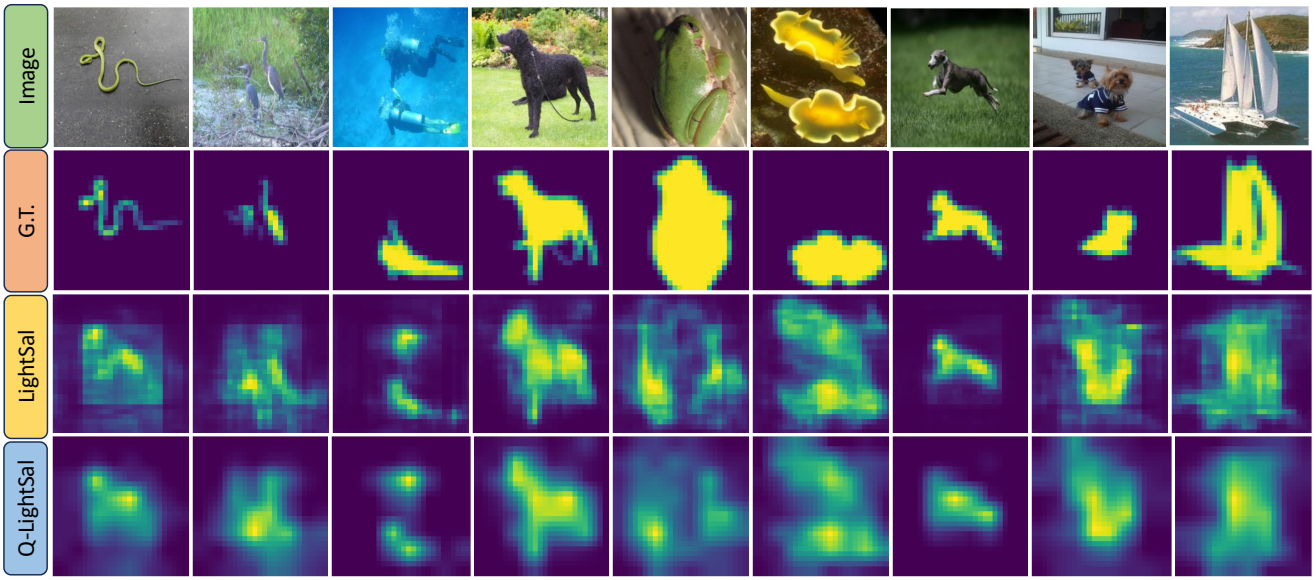


Figure 11: Comparing the input image, saliency ground truth (G. T.), `LimitNet`'s saliency output with and without quantization, denoted as Q-LightSal and LightSal, respectively. While the saliency detection branch does not achieve the same level of preciseness as the ground truth, it detects the parts of the image that include the target object, which bumps up the accuracy when we prioritize these sections in progressive offloading.

Table 5: Resource requirements and inference time of both `LimitNet`'s encoder and the saliency detector as float32 and quantized int8 models with and without DSP acceleration on nRF5340 and STM32F7 MCUs.

		nRF5340 MCU (Cortex-M33)				STM32F7 MCU (Cortex-M7)			
		without DSP accel.		with DSP accel.		without DSP accel.		with DSP accel.	
		int8	float32	int8	float32	int8	float32	int8	float32
Enc.	exe time (ms)	45,590	-	**1,969**	-	4,882	-	**237**	-
	RAM (KB)	344 (33%)	-	**344 (33%)**	-	**350 (17%)**	-	360 (17%)	-
	Flash (KB)	**94 (9%)**	-	100 (10%)	-	**91 (4%)**	-	107 (5%)	-
Sal.	exe time (ms)	4,972	1,046	**220**	981	522	576	**23**	655
	RAM (KB)	26 (2%)	99 (10%)	**26 (2%)**	99 (10%)	**24 (1%)**	99 (5%)	26 (1%)	99 (5%)
	Flash (KB)	**96 (9%)**	111 (11%)	102 (10%)	117 (11%)	**93 (4%)**	109 (5%)	107 (5%)	122 (6%)

on the nRF5340 MCU and compare it to JPEG ($Q = 50$). Fig. 12 shows that the encoding takes about 11% longer, i.e., about 220 ms longer, for `LimitNet` when compared to JPEG on the nRF. Also, on the STM, `LimitNet` encoding takes 260 ms while JPEG needs 249 ms, i.e., `LimitNet` requires about 4% more time than JPEG. Regarding energy consumption, `LimitNet` consumes 16.6 mJ, which is 0.65 mJ or 4% higher than JPEG ($Q = 50$) on nRF, see Fig. 12.

For this experiment, we use an implementation of JPEG that has been optimized for Cortex-M33 [1]; however, this implementation does not support progressive encoding while `LimitNet` does. To provide a progressive bitstream, ProgJPEG partitions the DCT coefficient table into multiple segments and encodes each separately.

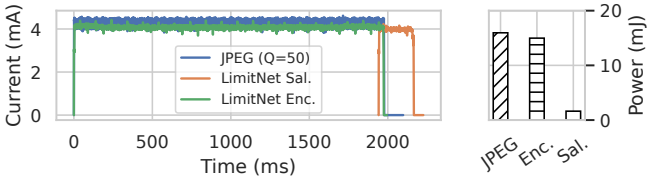


Figure 12: Energy and current consumption of `LimitNet`'s encoder and saliency detection compared to JPEG (for one predefined constant quality=50) on nRF5340 with $V = 1860mV$.

This entails quantization and the creation of Huffman tables, increasing the run-time of ProgJPEG over JPEG. Therefore, compared to `LimitNet`, we expect that ProgJPEG takes much more time[3].

[3]While some MCUs, e.g., STM, have a hardware-accelerated JPEG peripheral, others, e.g., nrf5340, do not have one. For a fair comparison, we opted to use an implementation that does not utilize such accelerators.

4.4.3 Limited Networking Scenarios. To assess the advantages of LimitNet and compare it to other models in real-world network scenarios, we conduct evaluations for LoRaWAN. In these experiments, we simulate a network with a gamma distribution under the bandwidth range of LoRaWAN [21], see Fig. 13. In each scenario, we analyze LimitNet's performance compared to Starfish [41] and ProgJPEG [97]. We select these two baselines as they are, to the best of our knowledge, the only image compression methods supporting progressive inference while being implementable on MCUs. Our results show that LimitNet exhibits a consistent progressive improvement in accuracy over time and outperforms baselines by a considerable margin; see Fig. 13. The main factor for this superior performance comes from LimitNet's ability to start reconstruction immediately after sending at most 40 bytes for the saliency map, whereas ProgJPEG needs to send at least the first scan, and Starfish needs to transmit a noticeable amount of data to reconstruct a meaningful image.

4.4.4 Impact of packet loss. As discussed in Section 2.2, LPWANs often have high packet loss rates, necessitating retransmission of dropped packets. As explained in Section 3.5.1, in the case of packet loss, LimitNet retransmits packets based on their importance, ensuring that the decoder receives the most important data first. ProgJPEG also sends data in multiple importance-based DCT scans. In Starfish, despite having regional importance distribution for packet loss resiliency, it has no packet prioritization and sends the data in a random order. In Table 6, we report the impact of different packet loss rates on LimitNet, ProgJPEG [97], and Starfish [41] during a single communication interval. Under different packet loss rates, although LimitNet, Starfish, and ProgJPEG have all sent an equal amount of data, LimitNet outperforms the other models by a significant margin due to its content-aware offloading regime.

Table 6: Top-1 accuracy of LimitNet, ProgJPEG, and Starfish on CIFAR100 with 2.5KB/s bandwidth and different Packet Loss Rates (PLR) after a single connection cycle (0.74-sec of transmission per minute). The first number shows the loss rate, and the second shows the received bytes after a cycle.

	10% PLR (1.62 KB)	40% PLR (1.31 KB)	70% PLR (0.5 KB)
LimitNet	**82.06%**	**81.2%**	**71.4%**
ProgJPEG	61.7%	43.1%	0%
Starfish	10.2%	2.6%	1%

5 RELATED WORK

The design of LimitNet draws inspiration from a wealth of prior research, including image compression, progressive image compression, compressive sensing, edge offloading, and importance detection methods:

Image compression: We categorize image compression models into two groups: classical approaches such as JPEG [97], JPEG2000 [90], JPEG-XR [4], WEBP [102], BPG [16] and recent deep learning-based methods, including Hyperprior [13], Full-Resolution [94], and others [12, 23, 77, 91]. While providing excellent performance, deep models and most of the classical compression models [4, 16, 90, 102] cannot be utilized on embedded MCUs due to their stark resource demands. In contrast, LimitNet employs a lightweight encoder that is executable on weak devices such as nRF5340 (Coretex M33) and STM32F7 (Cortex-M7).

Progressive image compression: Most classic compression models offer a progressive option [4, 16, 90, 102]. Among them, ProgJPEG [97] is the only one that can be deployed on resource-constrained embedded devices. Recently, learning-based compression models [28, 55, 68, 91, 104] have emerged, employing diverse techniques such as RNNs [34, 48, 93, 94], bit prioritization [46, 57], multiple decoders [18], and nested quantization [69] to produce progressive bitstreams. Similar to their non-progressive counterparts, these deep models are unsuitable for weak devices due to their high resource demands.

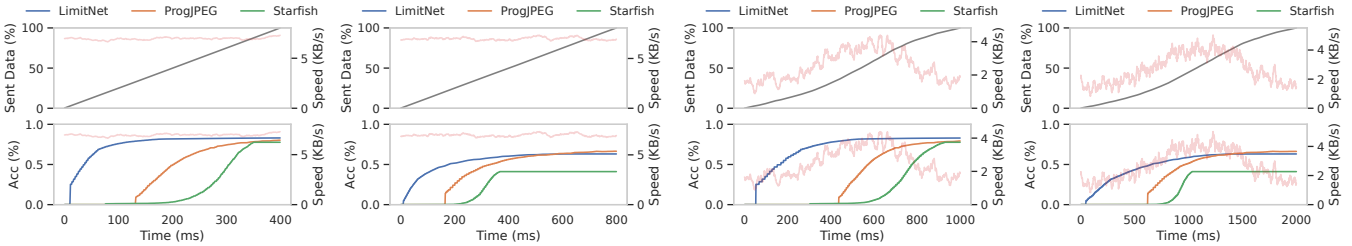
Edge offloading models: We categorize offloading models into two types: those with autoencoders [40, 42–45, 70, 71, 88, 103], and those without autoencoders [22, 30, 47, 58, 59, 80, 106]. While these models are usually focused on vision tasks, other models explore offloading for further tasks, including speech recognition and semantic segmentation [9, 24, 56, 72, 73, 75, 76]. However, in contrast to LimitNet, all of these models lack progressive offloading.

Importance Detection: We categorize importance detection models into three types: ROI detection [26, 35, 84, 85, 95], Explainable-AI [20, 33, 87, 99], and Saliency Detection [39, 60, 67, 100, 101, 107]. However, running these models on embedded devices is often infeasible due to their high computational and resource costs, comparable to or even greater than running a full classifier. In LimitNet we use a lightweight saliency detector, which is not as accurate as these models but can significantly improve the performance of progressive offloading when dealing with limited bandwidth.

6 DISCUSSION

Limitations: LimitNet generates a progressive bitstream, but this characteristic comes with a drawback as illustrated in Fig. 8. The progressive nature of the bitstream hinders LimitNet from achieving its maximum performance. The reason is that when only part of the data is available for decoding, the model expends effort in handling this incompleteness, degrading the maximum accuracy attainable.

ImageNet1000/COCO performance: As reported in Fig. 8a and Fig. 8c, the performance of LimitNet on ImageNet1000 and COCO is not as good as state-of-the-art. Nevertheless, we evaluated our model on ImageNet1000 to show the maximum capability. However, for real-world applications on weak devices, such as in the recent literature [41–44], the target task is usually much simpler, e.g., similar to CIFAR100. Furthermore, ImageNet1000's image sizes are too large (224×224 after preprocessing), each occupying around 150KB, which is a significant memory overhead for MCUs. Therefore, for such constrained devices, it is reasonable to opt for smaller input sizes.



(a) 6.25 KB/s - CIFAR100 (b) 6.25 KB/S - ImageNet1000 (c) Dynamic - CIFAR100 (d) Dynamic - ImageNet1000

Figure 13: `LimitNet` performance simulation under LoRaWAN [15] on ImageNet1000 [27] and CIFAR100 [51] datasets compared to Starfish [41] and ProgJPEG [97]. We evaluate the performances on each dataset under two different network settings: (a, b) with LoRaWAN's maximum bandwidth and (c, d) with a dynamic bandwidth. These plots show the average outcomes obtained from multiple experiments.

Other vision tasks: `LimitNet` incorporates a saliency detection mechanism. Therefore, it works on any vision task that relies mainly on the foreground of an image. Given `LimitNet`'s superior performance on CIFAR100 compared to ImageNet1000, we believe that `LimitNet` can yield promising results on less complex datasets like MNIST [54], SVHN [78] and Flowers [79]. On the other hand, our observations indicate that `LimitNet`'s performance diminishes on more challenging tasks such as object detection when precise focus on multiple objects is required, see Fig. 8.

Comparison to the SOTA: Starfish [41] uses dropout to add redundancy, which can deal with packet loss and is a first step towards progressiveness, as its accuracy does not collapse when receiving partial data. However, it is important to guide the model towards prioritizing the main objects, using saliency supported by gradual scoring (see Fig. 1) when transmitting. Starfish and ProgJPEG fail to do so, resulting in inferior performance (see Fig. 8). FLEET [42], on the other hand, has 4 fixed progressive steps, transmitting at least 2 KB of values, in each step. In contrast to the existing SOTA, `LimitNet` can progressively transmit the encoded data value by value in the order of importance, after initially transmitting the saliency map (at most 40 bytes) which results in a much more granular and flexible progressiveness.

Limitations of other alternatives: Pruning and quantization techniques fail to generate MCU-compatible models due to substantial performance degradation when pruning below 10%. Also, existing SOTA image compression models such as Ballé are too big for MCUs (165 times more parameters). Furthermore, despite the good performance of on-device inference models [64, 65], they are designed for specific network architecture and are not generalizable. Therefore, running a larger and better classifier on the cloud can be beneficial to achieve a higher accuracy. Additionally, with our design, it is possible to exchange the classifier on the server to either use an even better model or a model for a different task without needing to change anything on the MCU.

7 CONCLUSION

In this paper, we introduce `LimitNet`, an image compression and offloading model designed for extremely weak devices under LPWANs limitations: low bandwidth, short duty cycles, and high packet loss rates. We address these challenges by developing a lightweight content-aware progressive encoding scheme that prioritizes critical data during transmission based on their relative effects on classification accuracy. This progressive offloading allows the cloud to run the inference even with partial data availability, which is crucial when we have time-sensitive inferences such as deadlines or limited duty cycle budgets.

ACKNOWLEDGMENTS

We thank our anonymous reviewers and our shepherd, Yuanchun Li, for their insight and suggestions for improvement. This project has received funding from the Federal Ministry for Digital and Transport under the CAPTN-Fjörd 5G project grant no. 45FGU139_H , Federal Ministry for Economic Affairs and Climate Action under the Marispace-X project grant no. 68GX21002 and Marine Data Science (MarDATA) grant no. HIDSS-0005.

REFERENCES

[1] [n. d.]. JPEGEncoder4Cortex. https://github.com/noritsuna/JPEGEncoder4Cortex-M/tree/master.
[2] [n. d.]. TensorFlow Lite Micro. https://www.tensorflow.org/lite/micro. Accessed: [2023].
[3] [n. d.]. Zephyr Project RTOS. https://github.com/zephyrproject-rtos/zephyr. Accessed: [2023].
[4] 2009. JPEG XR Specification. https://jpeg.org/jpegxr/ Joint Photographic Experts Group (JPEG).
[5] 2023. *nRF5340 System-on-Chip.* https://www.nordicsemi.com/products/nrf5340
[6] 2023. *STM32F7 Series Microcontrollers.* https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html
[7] Mario Almeida, Stefanos Laskaridis, Stylianos I Venieris, Ilias Leontiadis, and Nicholas D Lane. 2022. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems* 21, 6 (2022), 1–24.
[8] Apple Inc. n.d.. *Use Emergency SOS via satellite on your iPhone.* https://support.apple.com/en-us/HT213426 Accessed: November 11, 2023.
[9] Juliano S Assine, Eduardo Valle, et al. 2021. Single-training collaborative object detectors adaptive to bandwidth and computation. *arXiv preprint arXiv:2105.00591* (2021).
[10] Kuldoshbay Avazov, An Eui Hyun, Alabdulwahab Abrar Sami S, Azizbek Khaitov, Akmalbek Bobomirzaevich Abdusalomov, and Young Im Cho. 2023.

Forest Fire Detection and Notification Method Based on AI and IoT Approaches. *Future Internet* 15, 2 (2023), 61.

[11] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. 2015. Density modeling of images using a generalized normalization transformation. *arXiv preprint arXiv:1511.06281* (2015).

[12] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. 2016. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704* (2016).

[13] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436* (2018).

[14] G Bjøtegaard. 2001. Calculation of average PSNR differences between RD-curves (vceg-m33). In *VCEG Meeting (ITU-T SG16 Q. 6), Austin, Texas, USA,, Tech. Rep. M*, Vol. 16090.

[15] Martin Bor, John Edward Vidler, and Utz Roedig. 2016. LoRa for the Internet of Things. (2016).

[16] BPG [n. d.]. BPG Image format. https://bellard.org/bpg/. Accessed: 2023-02-14.

[17] Chunlei Cai, Li Chen, Xiaoyun Zhang, and Zhiyong Gao. 2019. End-to-end optimized ROI image compression. *IEEE Transactions on Image Processing* 29 (2019), 3442–3457.

[18] Chunlei Cai, Li Chen, Xiaoyun Zhang, Guo Lu, and Zhiyong Gao. 2019. A novel deep progressive image compression framework. In *2019 Picture Coding Symposium (PCS)*. IEEE, 1–5.

[19] Luis Camal and Baris Aksanli. 2020. Building an energy-efficient ad-hoc network for wildlife observation. *Electronics* 9, 6 (2020), 984.

[20] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. 2020. Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks. *IEEE Transactions on Image Processing* 29 (2020), 4030–4043.

[21] Bharat S Chaudhari, Marco Zennaro, and Suresh Borkar. 2020. LPWAN technologies: Emerging application characteristics, requirements, and design considerations. *Future Internet* 12, 3 (2020), 46.

[22] Bo Chen, Zhisheng Yan, Hongpeng Guo, Zhe Yang, Ahmed Ali-Eldin, Prashant Shenoy, and Klara Nahrstedt. 2021. Deep contextualized compressive offloading for images. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 467–473.

[23] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. 2020. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7939–7948.

[24] Hyomin Choi, Robert A Cohen, and Ivan V Bajić. 2020. Back-and-forth prediction for deep tensor compression. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4467–4471.

[25] HeeSeok Choi, Heonchang Yu, and EunYoung Lee. 2019. Latency-classification-based deadline-aware task offloading algorithm in mobile edge computing environments. *Applied Sciences* 9, 21 (2019), 4696.

[26] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Advances in Neural Information Processing Systems*. 379–387.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[28] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. Drasic: Distributed recurrent autoencoder for scalable image compression. In *2020 Data Compression Conference (DCC)*. IEEE, 3–12.

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3156–3164.

[30] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 565–576.

[31] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. 2019. 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, 1–6.

[32] Fares Fourati and Mohamed-Slim Alouini. 2021. Artificial intelligence for satellite communication: A review. *Intelligent and Converged Networks* 2, 3 (2021), 213–243.

[33] Ruigang Fu, Qingyong Hu, Xiaohu Dong, Yulan Guo, Yinghui Gao, and Biao Li. 2020. Axiom-based Grad-CAM: Towards Accurate Visualization and Explanation of CNNs. *arXiv preprint arXiv:2008.02312* (2020).

[34] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. 2016. Towards conceptual compression. *Advances In Neural Information Processing Systems* 29 (2016).

[35] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2961–2969.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[37] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[38] Ali Hojjat, Janek Haberer, and Olaf Landsiedel. 2023. ProgDTD: Progressive Learned Image Compression With Double-Tail-Drop Training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 1130–1139.

[39] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. 2017. Deeply Supervised Salient Object Detection with Short Connections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[40] Diyi Hu and Bhaskar Krishnamachari. 2020. Fast and accurate streaming CNN inference via communication compression on the edge. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 157–163.

[41] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. 2020. Starfish: resilient image compression for aiot cameras. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 395–408.

[42] Jin Huang, Hui Guan, and Deepak Ganesan. 2023. Re-Thinking Computation Offload for Efficient Inference on IoT Devices with Duty-Cycled Radios. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking* (Madrid, Spain) *(ACM MobiCom '23)*. Association for Computing Machinery, New York, NY, USA, Article 13, 15 pages. https://doi.org/10.1145/3570361.3592514

[43] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. 2020. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–12.

[44] Kai Huang and Wei Gao. 2022. Real-time neural network inference on extremely weak devices: agile offloading with explainable AI. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 200–213.

[45] Mikolaj Jankowski, Deniz Gündüz, and Krystian Mikolajczyk. 2020. Joint device-edge inference over wireless links with pruning. In *2020 IEEE 21st international workshop on signal processing advances in wireless communications (SPAWC)*. IEEE, 1–5.

[46] Seungmin Jeon, Kwang Pyo Choi, Youngo Park, and Chang-Su Kim. 2023. Context-Based Trit-Plane Coding for Progressive Image Compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14348–14357.

[47] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation offloading for machine learning web apps in the edge server environment. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1492–1499.

[48] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. 2018. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4385–4393.

[49] Toshiaki Koike-Akino and Ye Wang. 2020. Stochastic bottleneck: Rateless auto-encoder for flexible dimensionality reduction. In *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2735–2740.

[50] Mark A Kramer. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal* 37, 2 (1991), 233–243.

[51] A. Krizhevsky. 2009. *CIFAR-100 (Canadian Institute for Advanced Research)*. Technical Report. University of Toronto. https://www.cs.toronto.edu/~kriz/cifar.html

[52] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–15.

[53] Alexandru Lavric, Adrian I Petrariu, and Valentin Popa. 2019. Sigfox communication protocol: The new era of iot?. In *2019 international conference on sensing and instrumentation in IoT Era (ISSI)*. IEEE, 1–4.

[54] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST Handwritten Digit Database.

[55] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack. 2018. Context-adaptive entropy model for end-to-end optimized image compression. *arXiv preprint arXiv:1809.10452* (2018).

[56] Joo Chan Lee, Yongwoo Kim, SungTae Moon, and Jong Hwan Ko. 2021. A splittable dnn-based object detector for edge-cloud collaborative real-time video inference. In *2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–8.

[57] Jae-Han Lee, Seungmin Jeon, Kwang Pyo Choi, Youngo Park, and Chang-Su Kim. 2022. DPICT: Deep progressive image compression using trit-planes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16113–16122.

[58] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. 2018. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 402–411.

[59] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[60] Jun Li, Yuhua Wang, Xin Qi, Xuming Hou, and Ming-Ming Liu. 2018. AMULET: Aggregated Multi-Level Feature U-Net for Salient Object Detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[61] Mu Li, Kede Ma, Jane You, David Zhang, and Wangmeng Zuo. 2020. Efficient and effective context-based convolutional entropy modeling for image compression. *IEEE Transactions on Image Processing* 29 (2020), 5900–5911.

[62] Weiping Li. 2001. Overview of fine granularity scalability in MPEG-4 video standard. *IEEE Transactions on circuits and systems for video technology* 11, 3 (2001), 301–317.

[63] Jongseong Lim, Sunghun Jung, Chan JeGal, Gwanghoon Jung, Jung Ho Yoo, Jin Kyu Gahm, and Giltae Song. 2022. LEQNet: Light Earthquake Deep Neural Network for Earthquake Detection and Phase Picking. *Frontiers in Earth Science* 10 (2022), 848237.

[64] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. 2021. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint arXiv:2110.15352* (2021).

[65] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.

[66] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[67] Xueyang Liu, Li Zhang, Lei Zhang, Yuhao Zhang, Wen Ma, and Tieniu Huang. 2018. PiCANet: Learning Pixel-wise Contextual Attention for Saliency Detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[68] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*. 10012–10022.

[69] Yadong Lu, Yinhao Zhu, Yang Yang, Amir Said, and Taco S Cohen. 2021. Progressive neural image compression with nested quantization and latent ordering. In *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 539–543.

[70] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. 2019. 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*. 21–26.

[71] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. 2020. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access* 8 (2020), 212177–212193.

[72] Yoshitomo Matsubara and Marco Levorato. 2020. Split computing for complex object detectors: Challenges and preliminary results. *arXiv preprint arXiv:2007.13312* (2020).

[73] Yoshitomo Matsubara and Marco Levorato. 2021. Neural compression and filtering for edge-assisted real-time object detection in challenged networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2272–2279.

[74] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2022. Split computing and early exiting for deep learning applications: Survey and research challenges. *Comput. Surveys* 55, 5 (2022), 1–30.

[75] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. 2022. SC2: Supervised compression for split computing. *arXiv preprint arXiv:2203.08875* (2022).

[76] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. 2022. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2685–2695.

[77] David Minnen, Johannes Ballé, and George D Toderici. 2018. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems* 31 (2018).

[78] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Street View House Numbers (SVHN) Dataset.

[79] Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes.

[80] Daniele Jahier Pagliari, Roberta Chiaro, Enrico Macii, and Massimo Poncino. 2020. Crime: Input-dependent collaborative inference for recurrent neural networks. *IEEE Trans. Comput.* 70, 10 (2020), 1626–1639.

[81] Xuebin Qin, Deng-Ping Fan, Chenyang Huang, Cyril Diagne, Zichen Zhang, Adrià Cabeza Sant'Anna, Albert Suarez, Martin Jagersand, and Ling Shao. 2021. Boundary-aware segmentation network for mobile and web applications. *arXiv preprint arXiv:2101.04704* (2021).

[82] Anitha Ramachandran and Arun Kumar Sangaiah. 2021. A review on object detection in unmanned aerial vehicle surveillance. *International Journal of Cognitive Computing in Engineering* 2 (2021), 215–228.

[83] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 779–788.

[84] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), 779–788.

[85] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 6 (2017), 1137–1149.

[86] Amir Said and William A Pearlman. 1996. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on circuits and systems for video technology* 6, 3 (1996), 243–250.

[87] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2020. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision* 128, 2 (2020), 336–359.

[88] Jiawei Shao and Jun Zhang. 2020. 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, 1–6.

[89] Axel Sikora, Manuel Schappacher, Zubair Amjad, et al. 2019. Test and measurement of LPWAN and cellular IoT networks in a unified testbed. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Vol. 1. IEEE, 1521–1527.

[90] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi. 2001. The JPEG 2000 still image compression standard. *IEEE Signal processing magazine* 18, 5 (2001), 36–58.

[91] Rige Su, Zhengxue Cheng, Heming Sun, and Jiro Katto. 2020. Scalable learned image compression with a recurrent neural networks-based hyperprior. In *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 3369–3373.

[92] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[93] George Toderici, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. 2015. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085* (2015).

[94] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. 2017. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 5306–5314.

[95] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. 2013. Selective Search for Object Recognition. In *International Journal of Computer Vision*, Vol. 104. Springer, 154–171.

[96] Alpha Vijayan, B Meenaskshi, Aditya Pandey, Akshat Patel, and Arohi Jain. 2022. Video anomaly detection in surveillance cameras. In *2022 International Conference for Advancement in Technology (ICONAT)*. IEEE, 1–4.

[97] Gregory K Wallace. 1991. The JPEG still picture compression standard. *Commun. ACM* 34, 4 (1991), 30–44.

[98] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.

[99] Haofan Wang, Zifan Wang, Mengnan Du, Fan Yang, Zijian Zhang, Sirui Ding, Piotr Mardziel, Xia Hu, and Xia Hu. 2021. Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 11 (2021), 5185–5198.

[100] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, and Baocai Yin. 2019. Bilateral Multi-Pooling with Bi-Directional Message Passing for Salient Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[101] Xinyu Wang, Xinyuan Cao, Chunhua Shen, and Liang Lin. 2018. Recurrent Residual Module for Fast Inference in Videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[102] WebP [n. d.]. WebP. https://developers.google.com/speed/webp/docs/compression. Accessed: 2023-02-14.

[103] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. 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*.

476–488.

[104] Juncheol Ye, Hyunho Yeo, Jinwoo Park, and Dongsu Han. 2023. AccelIR: Task-Aware Image Compression for Accelerating Neural Restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18216–18226.

[105] Tayyaba Zainab, Jens Karstens, and Olaf Landsiedel. 2023. LightEQ: On-Device Earthquake Detection with Embedded Machine Learning. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*. 130–143.

[106] Liekang Zeng, En Li, Zhi Zhou, and Xu Chen. 2019. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things. *IEEE Network* 33, 5 (2019), 96–103.

[107] Dong Zhang, Yuchen Han, Ming-Hsuan Yang, Tong Zhang, Wei Liu, Xiaokang Yang, and Jing Guo. 2018. Nested Lateral Descriptive Features for Salient Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.