Deep Contextualized Compressive Offloading for Images

Bo Chen[1], Zhisheng Yan[2], Hongpeng Guo[1], Zhe Yang[1]
Ahmed Ali-Eldin[3], Prashant Shenoy[4], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign [2]George Mason University
[3]Chalmers University of Technology and UMass [4]Amherst University of Massachusetts Amherst
[1]{boc2, hg5, zheyang3, klara}@illinois.edu, [2]zyan4@gmu.edu, {[3]ahmeda, [4]shenoy}@cs.umass.edu

ABSTRACT

Recent years have witnessed sensors becoming an indispensable part of our life with the camera being one of the most popular and widely deployed sensors. The camera gives rise to numerous vision-based IoT applications that generate high-level understandings of a live video stream by performing analysis on end devices like mobile or embedded devices. Typically, these applications are built with deep learning (DL) models to conduct complex vision tasks, e.g., image classification and object detection. Due to the prohibitive cost of running DL models on end devices close to the camera and with limited computation capabilities, it is widely adopted to offload the computation to a nearby powerful edge server. However, there is a gap between the restricted offloading bandwidth of the end device and the large volume of image data incurred by the live video stream. In this paper, we present Deep Contextualized Compressive Offloading for Images (DCCOI), a lightweight, context-aware, and bandwidth-efficient offloading framework for images. DCCOI consists of the spatial-adaptive encoder, a lightweight neural network, to spatial-adaptively compress the image, and the generative decoder for reconstructing the image from the compressed data. In contrast to existing DL-based encoders, the spatial-adaptive encoder allows an image region to be encoded into different numbers of feature values based on the information in it. This offers a variable-length coding method for image compression, which is a more optimal way for compression than the fix-length coding method took by existing DL-based compression approaches and demonstrates superior accuracy-compression rate trade-offs. We evaluate DCCOI against several baseline compression techniques while serving an object detection-based application. The results show that DCCOI roughly reduces the offloading size of JPEG by a factor of 9 and DeepCOD, the state-of-the-art offloading approach, by 20% with similar accuracy and a compression overhead less than 50ms.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Machine learning approaches**;

AIChallengeIoT 21, November 15–17, 2021, Coimbra, Portugal

ACM ISBN 978-1-4503-9097-2/21/11…$15.00
https://doi.org/10.1145/3485730.3493452

KEYWORDS

Deep Learning, Image Compression, Internet of Things

ACM Reference Format:
Bo Chen[1], Zhisheng Yan[2], Hongpeng Guo[1], Zhe Yang[1], Ahmed Ali-Eldin[3], Prashant Shenoy[4], Klara Nahrstedt[1] . 2021. Deep Contextualized Compressive Offloading for Images . In *The 3rd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT 21), November 15–17, 2021, Coimbra, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3485730.3493452

1 INTRODUCTION

Vision-based IoT applications (vision apps) are becoming pervasive nowadays due to the advancement in deep learning (DL) [8] and the strong sensing capabilities of widely deployed cameras. Vision apps process live-captured videos with sophisticated DL models to generate a high-level understanding of the image/video. Examples of vision apps include human face recognition [18], pedestrian detection [4], or traffic monitoring [14] applications. However, the vision app is usually demanded on a near-camera end device with limited computation capabilities, e.g., mobile IoT devices [21], embedded devices [14], and underwater sensor nodes [13], which can hardly run large DL models. To facilitate vision apps on end devices, it is widely adopted to offload the computation to an edge server, which executes all or a part of the DL model. Due to the real-time nature of vision apps, a large volume of images has to be offloaded, which conflicts with the limited offloading bandwidth.

To address the bandwidth issue in offloading, a new category of compression techniques, *machine-centric image compression* [6], has been proposed. Unlike traditional compression standards, e.g., JPEG [25], and JPEG2000 [23], that preserve the visual quality of images, the machine-centric image compression technique aims at maintaining the performance metrics when conducting image classification or object detection, e.g., the top-1 accuracy and the mean average precision (mAP) [7] Machine-centric image compression can be categorized into server-assisted compression [6, 16, 17] and DL-based compression [2, 19, 26].

In server-assisted compression techniques, the end device adaptively compresses different image regions based on the *context* sent from the edge server. The context is defined as the indicator of whether a particular compression technique should be applied regarding a vision app (IoC) in an image region. For instance, the regions of interest (ROI) of images processed at the edge server have been treated as the context to allow selected image regions to be transmitted from the end device at a higher resolution [6, 16, 17]. One limitation of this approach is its reliance on server-side feedback, which makes its effectiveness dependent on the network latency. Specifically, if the ROI sent by the edge server is delayed

and deviates from the actual ROI on the end device, server-assisted compression becomes ineffective in offloading for images.

DL-based compression techniques do not depend on the network latency like server-assisted compression. They train an encoder network on the end device to extract features from raw images and a decoder network on the edge server to reconstruct the image from extracted features. DL-based compression techniques like DeepCOD [26] have been successful at maintaining the accuracy of vision apps with a much lower compression rate (referred to as "rate") than traditional compression techniques, i.e., higher *compression effectiveness*. The impediment of existing DL-based compression is that their encoder is inherently a fixed-length coding method. It sequentially applies one or multiple convolutional neural networks (CNN) to an image and encodes heterogeneous image contents in different regions with the same number of feature values. However, according to Shannon's source coding theorem [22], image contents containing less information should be encoded with shorter coding lengths, which makes existing DL-based compression techniques suboptimal (*encoding suboptimality*). The fundamental problem of DL-based compression techniques is that they lack the *spatial-adaptability* of server-assisted compression techniques.

We present Deep Contextualized Compressive Offloading for Images (DCCOI), a lightweight, context-aware, and bandwidth-efficient offloading framework for images. DCCOI significantly reduces the offloading bandwidth when compared to existing compression techniques with similar accuracy and a small compression overhead, i.e., the latency of compression. We introduce *tightly coupled context extraction* and *hierarchical masked filtering* in the design of a spatial-adaptive encoder, a lightweight neural network. Tightly coupled context extraction embeds a context extractor network in the spatial-adaptive encoder, which enables the context extractor to be learned regarding the compression technique and the vision app. Thus, the context is extracted to reflect the IoC correctly. Hierarchical masked filtering constructs a hierarchy of filters allowing different image regions to be adaptively compressed based on the context. It overcomes the encoding suboptimality of existing DL-based compression techniques as a variable-length coding method. A generative decoder is also built to reconstruct the image from adaptively compressed data based on a generative network. Furthermore, the spatial-adaptive encoder and the generative decoder can be trained end-to-end to optimize the accuracy of the vision app and the compression rate.

We evaluate DCCOI against several compression baselines while serving an object detection-based application. It is shown that DCCOI roughly reduces the offloading size of JPEG by a factor of 9 and DeepCOD [26], the state-of-the-art offloading approach, by 20% with similar accuracy and a compression overhead less than 50ms.

The contribution of this paper is summarized as follows.

- We present DCCOI, a lightweight, context-aware, and bandwidth-efficient offloading framework for images consisting of a spatial-adaptive encoder and a generative decoder.
- We design the spatial-adaptive encoder, a lightweight neural network based on tightly coupled context extraction and hierarchical masked filtering, which allow the context map to be extracted correctly and drives spatial-adaptive compression of the image.
- We evaluate DCCOI against several baselines while serving an object detection-based application. DCCOI demonstrates superior performance in saving the bandwidth while maintaining similar accuracy with a small compression overhead.

2 RELATED WORK

2.1 Traditional Image Compression

Typical traditional image compression techniques include JPEG [25], JPEG2000 [23] and WebP [9]. JPEG divides the image into 8×8 macroblocks and operates on the YUV components of them. It mainly consists of three steps: 1) discrete cosine transform (DCT): extract DCT coefficients from the YUV components, 2) quantization: divide DCT coefficients in all macroblocks by a quantization table and round results to integers, and 3) entropy encoding: apply Huffman coding to the quantized DCT coefficients. WebP is similar to JPEG in the sense that it also operates on macroblocks and involves DCT, quantization, and entropy encoding. WebP improves on JPEG via predictive coding that utilizes information in neighboring macroblocks to predict a macroblock. Unlike these techniques that focus on visual quality, DCCOI focuses on optimizing accuracy regarding vision apps and minimizing the offloading size.

2.2 Machine-Centric Image Compression

Server-assisted compression. Server-assisted compression exploits server-side feedback to assist compression. [6, 16, 17] propose to exploit the server-side ROI information to drive spatial quality adaptation at the client. [5] tracks objects on the mobile device by utilizing the object recognition result computed on the server. The limitation of server-assisted compression is that the unpredictable network latency can hamper its performance. In contrast, DCCOI does not rely on server-side feedback to function.

DL-based compression. The autoencoders [2, 19, 20, 24] are a type of DL-based compression that builds symmetric deep encoder and decoder networks to compress and reconstruct the image, respectively. They are able to compress images into a much smaller size than traditional compression techniques, e.g., JPEG. However, their encoding side demands sophisticated models to extract latent features from the image, which places a huge burden on end devices with limited computation capabilities. To deal with this problem, DeepCOD [26] proposes an "imbalanced" autoencoder. The limitation of DeepCOD is that it inherently treats image regions containing different amounts of information in the same way, which is sub-optimal. DCCOI introduces the spatial-adaptive encoder that adaptively compresses the image based on the extracted context map and greatly improves the compression rate of DeepCOD with almost no loss in accuracy.

3 DEEP CONTEXTUALIZED COMPRESSIVE OFFLOADING FOR IMAGES

3.1 Overview

The design of DCCOI is shown in Figure 1, which mainly consists of a spatial-adaptive encoder (SA-ENC) and a generative decoder (G-DEC). On the end device, the image is processed by the spatial-adaptive encoder into the compressed data. The compressed data on the end device is transmitted over wireless networks to the edge

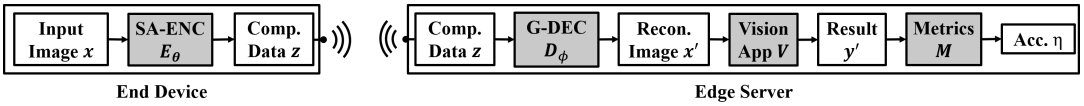

Figure 1: System Architecture

server. On the edge server, the compressed data is processed by the generative decoder to reconstruct the image. The reconstructed image will be used to generate predictive results and evaluate accuracy.

3.2 Problem Formulation

The spatial-adaptive encoding process compresses the input image into the compressed data, which is formulated in Equation 1.

$$z = E_\theta(x), \tag{1}$$

where x represents the input image, z represents the compressed data, and θ is the parameter of the spatial-adaptive encoder. The generative decoding process reconstructs the image from the compressed data, which is formulated in Equation 2.

$$x^{'} = D_\phi(z) = D_\phi(E_\theta(x)), \tag{2}$$

where $x^{'}$ represents the reconstructed image data and ϕ represents the parameter of the generative decoder. The vision app makes predictions, e.g., object detection, based on the reconstructed image data, which can be formulated as:

$$y^{'} = V(x^{'}), \tag{3}$$

where V abstracts the prediction of the vision app and $y^{'}$ is the prediction result, e.g., detected bounding boxes, of reconstructed image data $x^{'}$. The prediction result is used to evaluate accuracy regarding the vision app, which is formulated as:

$$\eta = M(y^{'}), \tag{4}$$

where M is an abstraction for metrics like the top-1 accuracy and the mAP. The compression rate r is used to evaluate the compression effectiveness, which is defined in Equation 5.

$$r = |z|/|x|, \tag{5}$$

where $|\cdot|$ denotes the size of the data.

Our goal is to find parameters, θ and ϕ, that maximize both accuracy and the compression rate, which can be formulated into a multi-objective optimization (MOO) problem as in Equation 6.

$$\max_{\theta,\phi} \boldsymbol{f}(\theta,\phi) = (\eta, r) \subset \mathbb{R}^2, \tag{6}$$

4 SPATIAL-ADAPTIVE ENCODER

The spatial-adaptive encoder is a lightweight neural network designed with tightly coupled context extraction and hierarchical masked filtering. As shown in Figure 2, it mainly consists of a context extractor, a spatial-adaptive feature extractor, a quantization module, and a lossless encoding module.

4.1 Context Extractor

The context extractor is implemented with CNN to generate context maps $m_i \subset \mathbb{R}^{3\times h_i\times w_i}$, $i = 1, ..., L$, from the input image $x \subset \mathbb{R}^{3\times h\times w}$. h and w are the height and width of the input image, respectively. $h_i = \frac{h}{2^{L-i+3}}$ and $w_i = \frac{w}{2^{L-i+3}}$ are the height and width of the i-th context map, respectively. The i-th context map m_i assumes the image x is partitioned into $h_i \times w_i$ equal-sized blocks. Each element in m_i indicates the IoC of the corresponding block.

4.2 Spatial-Adaptive Feature Extractor

Spatial-adaptive feature extractor adaptively encodes information in the image based on the context maps m_i, $i = 1, ..., L$, as shown in Figure 3. The key components towards spatial adaptability are a hierarchy of learnable filters (referred to as the filter) and the filtering masks (referred to as the mask).

The learnable filter is the basic operation to compress the image. We implement the filter $\mathcal{F}_i$, $i = 1, ..., L+1$, as a single-layer CNN whose input and output channels are the same [1]. We adopt a cascaded design of filters as shown at the top of Figure 3. The first filter $\mathcal{F}_1$ has its size and stride both set to 4×4. The size and stride of subsequent filters $\mathcal{F}_i$, $i = 2, ..., L+1$, are both 2×2. The rationale behind this setup of filters is to allow the image to be compressed with compression rates of $1:4^2$, $1:8^2$, ... Generally, the more filters an image region is processed with, information in the region is more compactly compressed.

The filtering mask indicates the image region that will be compressed with a specific sequence of filters. In other words, different masks control image compression with different compression rates. The mask $\hat{m}_i \subset \mathbb{R}^{3\times h\times w}$, $i = 1, ..., L+1$, is implemented as a boolean tensor derived from context maps. We derive the mask via logical operations as shown in Equation 7.

$$\hat{m}_i = \begin{cases} \dot{m}_i < \delta & i = 1 \\ (\neg\hat{m}_1) \bigwedge ... \bigwedge (\neg\hat{m}_{i-1}) \bigwedge (\dot{m}_i < \delta) & i = 2, .., L \\ (\neg\hat{m}_1) \bigwedge ... \bigwedge (\neg\hat{m}_{i-1}) & i = L+1 \end{cases} \tag{7}$$

where

$$\dot{m}_i = upsample(m_i, L - i + 3) \tag{8}$$

up-samples the context map m_i to the size $h_i \cdot 2^{L-i+3} \times w_i \cdot 2^{L-i+3} = h \times w$ with the nearest neighbor principle, which makes all boolean operations performed with the same size ($h \times w$). $\dot{m}_i < \delta$ maps each value in a tensor to a boolean value depending on whether that value is less than the threshold δ, which is set to 0.5 by default. The operator $\neg$ flips each boolean value in a boolean tensor. The operator $\bigwedge$ applies the logical AND operation to two boolean tensors.

Spatial-adaptive feature extraction is performed by processing image regions masked by $\hat{m}_i$ with filters $\mathcal{F}_i$, $i = 1, ..., L+2-i$, sequentially. In this way, we can derive the filtered feature v_i from $\hat{m}_i$ as formulated in Equation 9.

$$v_i = flatten(\mathcal{F}_{L+2-i}(...(\mathcal{F}_1(x))), \ddot{m}_i), i = 1, ..., L+1. \tag{9}$$

where

$$\ddot{m}_i = downsample(\hat{m}_i, L - i + 3) \tag{10}$$

down-samples the mask $\hat{m}_i$ to the size $\frac{h}{2^{L-i+3}} \times \frac{w}{2^{L-i+3}} = h_i \times w_i$, which changes the size of the mask to the size of the output of the

[1] Since the number of channels (3) does not change during compression, we omit it when describing the size of tensors generated during compression for simplicity.

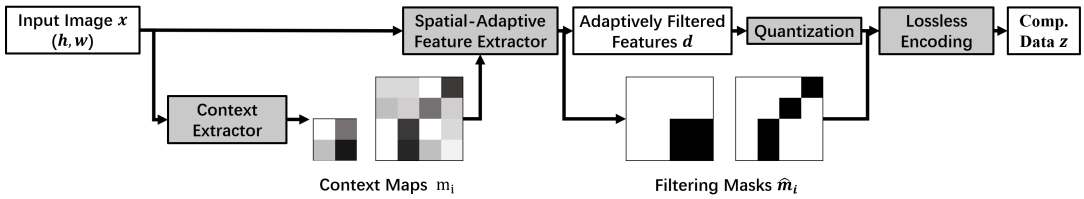

Figure 2: Spatial-Adaptive Encoder

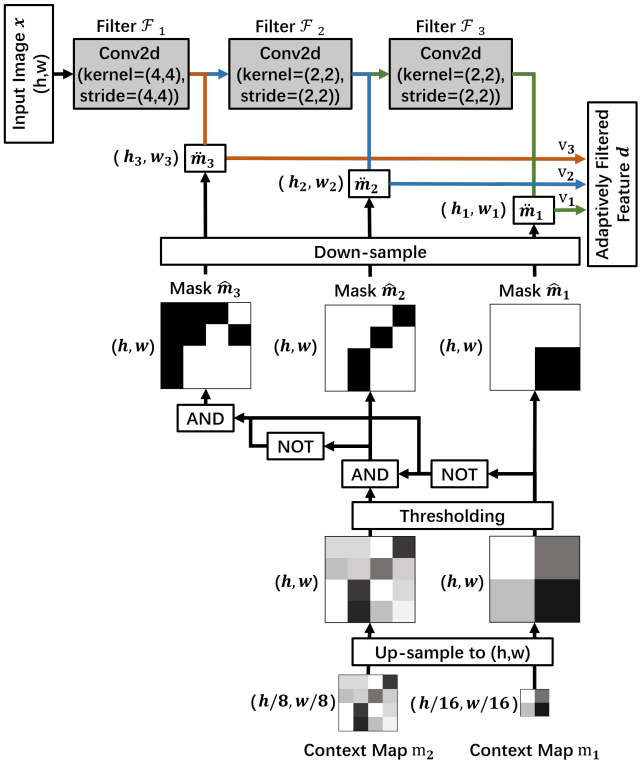

Figure 3: Spatial-Adaptive Feature Extractor

filter $\mathcal{F}_{L+2-i}$. $\mathcal{F}_k(\cdot)$, $k = 1, .., L+1-i$, applies the filter $\mathcal{F}_k$ to the feature. $flatten(a, b)$ flattens the feature a to a vector based on the mask b. A smaller index of the mask corresponds to more filters and hence more compact compression. As shown in Figure 3, the mask $\hat{m}_1$ indicates the bottom-right $\frac{h}{2} \times \frac{w}{2}$ block in the input image will be processed by filters $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$.

The adaptively filtered feature d is the concatenation of all filtered features:

$$d = v_1 \parallel ... \parallel v_{L+1} = (d_1, ..., d_N), \tag{11}$$

where N is the length of d.

4.3 Quantization & Lossless Encoding

We adopt the learning-based quantization technique [1] in the quantization module. It maps each value in the adaptively filtered feature d to one value in a set of quantization centers $C = \{c_1, ..., c_K\} \subset \mathbb{R}$. The lossless encoding module applies run-length encoding (RLE) and Huffman coding to the quantized feature $\hat{d} = (\hat{d}_1, ..., \hat{d}_N)$ and the filtering masks $\hat{m}_i$, $i = 1, ..., L$, to reduce the data size.

5 GENERATIVE DECODER

The generative decoder consists of the lossless decoding module, the adaptive up-sampling module, and the generative network as shown in Figure 4.

5.1 Lossless Decoding

The lossless decoding module decodes the compressed data z to recover the quantized features $\hat{d}$ and the masks $\hat{m}_i$, $i = 1, ..., L$ (the $(L+1)$-th mask can be derived from previous L masks). It technically reverses the operations of Huffman coding and RLE.

5.2 Adaptive Up-Sampling

The adaptive up-sampling module processes the quantized features $\hat{d}$ to obtain the feature $\hat{g} \subset \mathbb{R}^{3 \times \frac{h}{4} \times \frac{w}{4}}$, which will be used as the input to the generative network. The adaptive up-sampling module is implemented by up-sampling $\hat{d}$ based on the masks $\hat{m}_i$, $i = 1, ..., L+1$, with the nearest neighbor principle. The restored features $\hat{g}$ can be represented by Equation 12.

$$\hat{g} = g^{(L+1)} \odot \hat{m}_1 + ... + g^{(1)} \odot \hat{m}_{L+1}, \tag{12}$$

where $g^{(i)}$ denotes the feature map that is processed by a sequence of filters $\mathcal{F}_k$, $k = 1, ..., i$, quantized, and then up-sampled to the size $\frac{h}{4} \times \frac{w}{4}$ with the nearest neighbor principle:

$$g^{(i)} = upsample(Q(\mathcal{F}_i(...(\mathcal{F}_1(g)))), i), i = 1, ..., L+1, \tag{13}$$

where Q denotes the quantization step described in Section4.3.

To better learn parameters of DCCOI, we replace Equation 12 with the differentiable soft filter function as shown in Equation 14 during backward propagation.

$$\begin{aligned} \hat{g} = {} & g^{(L+1)} \odot \dot{m}_1 \\ & + g^{(L)} \odot (1 - \dot{m}_1) \odot \dot{m}_2 \\ & + ... \\ & + g^{(2)} \odot (1 - \dot{m}_1) \odot ... \odot (1 - \dot{m}_{L-1}) \odot \dot{m}_L \\ & + g^{(1)} \odot (1 - \dot{m}_1) \odot ... \odot (1 - \dot{m}_L), \end{aligned} \tag{14}$$

where $\dot{m}_i$, $i = 1, ..., L$, represents the context map m_i up-sampled to the size $h \times w$ (Equation 8).

5.3 Generative Network

The generative network computes the reconstructed image x' using the restored image feature $\hat{g}$. It is composed of the self-attention layer, the residual transposed convolution layer, and the single-layer convolution as illustrated in Figure 4. The self-attention layer is added for the enhancement of the spatial dependency during the

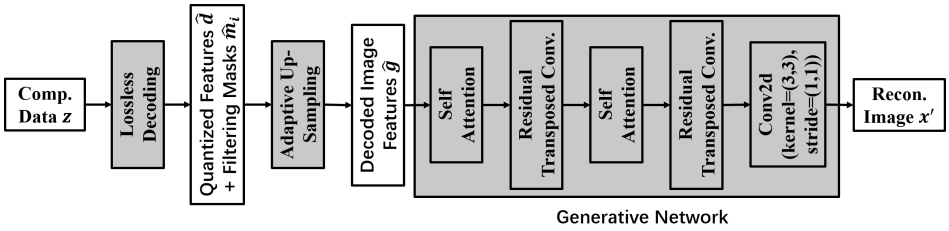

Figure 4: Generative Decoder

reconstruction of the image [27]. The residual transposed convolution layer expands the dimension of the input feature map to the original size of the image [3, 11].

6 IMPLEMENTATION

The loss function of the DCCOI model consists of the reconstruction loss, the pooling loss, the entropy penalty, and the orthogonal regularizor as shown in Equation 15.

$$\mathcal{L}_{DCCOI} = \mathcal{L}_{Recon} + \gamma_1 \mathcal{L}_{Filter} + \gamma_2 \mathcal{L}_E + \gamma_3 \mathcal{L}_{Orth}. \quad (15)$$

The reconstruction loss aims at distilling knowledge from the vision app V and making the reconstructed image behave in a similar way as the original image does regarding the vision app. We denote the output of the i-th layer of the model of the vision app V as $V^{(i)}(x)$, $i = 1, ..., L_V$, where x is the input of the network and L_V is the total number of layers of V. The reconstruction loss is formulated as in Equation 16.

$$\mathcal{L}_{Recon} = \sum_{i \in S} \left\| V^{(i)}(x) - V^{(i)}(x') \right\|_2, \quad (16)$$

where S is the set of selected layers. We select the input layer and the output layers of all Cross Stage Partial (CSP) bottlenecks in the YOLOv5 network to supervise the training.

The filter loss aims at suppressing the size of spatially filtered features. Intuitively, the filter loss can be simply represented by the size of spatially filtered features using the masks $\hat{m}_i$, $i = 1, ..., L+1$:

$$\mathcal{L}_{Filter} = \frac{1}{4^L} \oplus \hat{m}_1 + ... + \frac{1}{4^1} \oplus \hat{m}_L + \oplus \hat{m}_{L+1}, \quad (17)$$

where $\oplus m$ represents the sum of all elements in the matrix m and $\odot$ represents element-wise multiplication. The coefficient $\frac{1}{2^{2\times L}}$ accommodates the fact that different masks are responsible for compression with different comression rates.

To better learn parameters of DCCOI, we rewrite Equation 17 as the differentiable soft filter loss in Equation 18 during backward propagation.

$$\begin{aligned} \mathcal{L}_{Filter} = & \frac{1}{4^L} \oplus \dot{m}_1 \\ & + \frac{1}{4^{L-1}} \oplus ((1 - \dot{m}_1) \odot \dot{m}_2) \\ & + ... \\ & + \frac{1}{4^1} \oplus ((1 - \dot{m}_1) \odot ... \odot (1 - \dot{m}_{L-1}) \odot \dot{m}_L) \\ & + \oplus((1 - \dot{m}_1) \odot ... \odot (1 - \dot{m}_L)), \end{aligned} \quad (18)$$

where $\oplus m$ represents the sum of all elements in the matrix m and $\dot{m}_i$, $i = 1, ..., L$, represents the context map m_i up-sampled to the size $h \times w$ (Equation 8).

The entropy penalty suppresses the average information in quantized feature values $\hat{d}$ for better performance in entropy encoding. Lower average information means the entropy coding can compress the quantized feature values more compactly. We borrow the concept of the Shannon entropy[22] to evaluate the average information and formulate the entropy penalty in Equation 19.

$$\mathcal{L}_E = -\sum_{k=1}^{K} p_k \log p_k, \quad (19)$$

where K is the number of quantization centers and p_i represents the probability of a feature value being quantized to the i-th quantization center. To allow the entropy loss to be backward propagated, we represent the probability with the softmax function as shown in Equation 20.

$$p_k = \frac{1}{N} \sum_{i=1}^{N} \frac{exp(-\|d_i - c_k\|_2)}{\sum_{j=1}^{K} exp(-\left\|d_i - c_j\right\|_2)}, \quad (20)$$

where d_i, $i = 1, ..., N$, is the i-th feature value and c_j, $j = 1, ..., K$, is the j-th quantization center.

The orthogonal penalty is added to let the encoder E_θ meet the Set-Restricted Eigenvalue Condition, which ensures that the image can be recovered from the compressed data [26]. The orthogonal penalty is formulated in Equation 21.

$$\mathcal{L}_{Orth} = \left\| E_\theta'^T E_\theta' - I \right\|_2, \quad (21)$$

where $E_\theta' \in \mathbb{R}^{h_e \cdot w_e \cdot c_i \times c_o}$ is a matrix transformed from the convolution kernel of the encoder $E_\theta \in \mathbb{R}^{h_e \times w_e \times c_i \times c_o}$. I is the identity matrix.

7 EVALUATION

7.1 Methodology

Application. We conduct the evaluation on an object detection-based application. In this application, we apply YOLOv5 [12] to detect objects in the COCO 2017 dataset [15]. The performance metric we adopt is the mean average precision (mAP) [7].

Hardware. The spatial-adaptive encoder is implemented on two end devices: Raspberry Pi 4 Model B (RPi) and NVIDIA Jetson Nano (Nano). Raspberry Pi 4 Model B is equipped with a Quad-core Cortex-A72 CPU @ 1.5GHz. NVIDIA Jetson Nano is equipped with a Quad-core Cortex-A57 CPU @ 1.5GHz and a 128-core NVIDIA Maxwell architecture-based GPU. The generative decoder and the vision application are implemented on two Linux desktops as edge servers. One Linux desktop (RTX) is equipped with an Intel Core i7-9700K CPU @ 3.60GHz and two NVIDIA GeForce RTX 2080 Ti GPU. The other Linux desktop (GTX) is equipped with an Intel

Core i9-8950HK CPU @ 2.90GHz and one NVIDIA GeForce GTX 1080 GPU. The end device is connected to the Internet via WiFi or LTE, as detailed below. The edge server is connected to the campus network via a 1Gbps cable.

Networking. We consider two network conditions in the evaluation: WiFi and LTE. For WiFi, we adopt the 802.11ac standard with a frequency of 5 GHz and a bandwidth of 450Mbps. For LTE, we choose 4G LTE with an upload bandwidth of 50Mbps.

Baselines. 1) JPEG [10]: the de facto standard for image compression. 2) WebP [9]: an image format that is optimized to create smaller images on the website. 3) Intp: an approach that performs encoding by evenly down-sampling the image data before transmission on the end device. The image is decoded via the bilinear interpolation method on the edge server. 4) DeepCOD [26]: the state-of-the-art offloading approach that compresses the image via a single-layer CNN, a quantization module, and an entropy encoding module. The image is reconstructed using a generative model.

7.2 Accuracy-Rate Trade-off

Figure 5 demonstrates the accuracy-rate trade-offs of DCCOI and baselines in object detection, where each point represents the compression rate and the mAP of a compression configuration. Only meaningful configurations, whose accuracy is greater than that of DeepCOD, are kept in Figure 5. Among JPEG, WebP, and Intp, WebP achieves the best accuracy-rate trade-off. Specifically, WebP achieves a compression rate of 0.024 with an mAP of 0.5, which is slightly better than the compression rate of JPEG (0.036) at a similar mAP of 0.502 and significantly better than Intp. DCCOI and DeepCOD both greatly improve the accuracy-rate trade-off of others. DeepCOD achieves an mAP of 0.499 with a compression rate of 0.00489 while DCCOI achieves an mAP of 0.502 with a compression rate of only 0.00390. DCCOI roughly reduces the offloading size of JPEG and WebP by a factor of 9 and 6 with similar accuracy, respectively. In contrast to DeepCOD, DCCOI reduces the offloading size by 20.2% with similar accuracy.

To summarize, DL-based compression techniques achieve much better accuracy-rate trade-offs than other approaches. DCCOI outperforms DeepCOD by a roughly 20% reduction in the offloading size with similar accuracy.

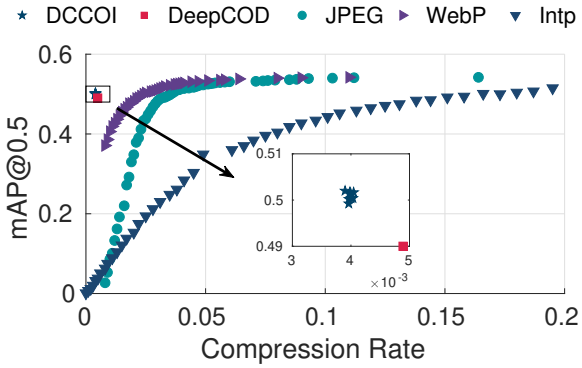

Figure 5: Accuracy-compression rate trade-offs (YOLOv5).

7.3 End-to-end Analysis

Based on the choices of the end device, i.e., RPi and Nano, and the edge server, i.e., GTX and RTX, we build four hardware architectures integrating different end devices and edge servers, which are denoted by RPi GTX, RPi RTX, Nano GTX, and Nano RTX. For the fair comparison of different approaches, we configure all approaches to produce similar accuracy with the maximum difference in mAP being only 0.003.

Figure 6 and Figure 7 present the end-to-end offloading latency using WiFi and LTE, respectively. When using WiFi, DeepCOD achieves the lowest end-to-end latency while the end-to-end latency of DCCOI is close to that of DeepCOD. When using LTE, DCCOI consistently achieves the lowest end-to-end latency, which reduces the latency of DeepCOD by roughly 40% and reduces the latency of JPEG by approximately a factor of 5. The compression overhead (Enc) of DCCOI is consistently lower than 50ms.

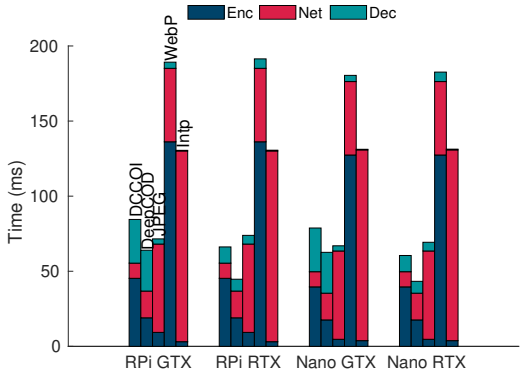

Figure 6: End-to-end offloading latency using WiFi

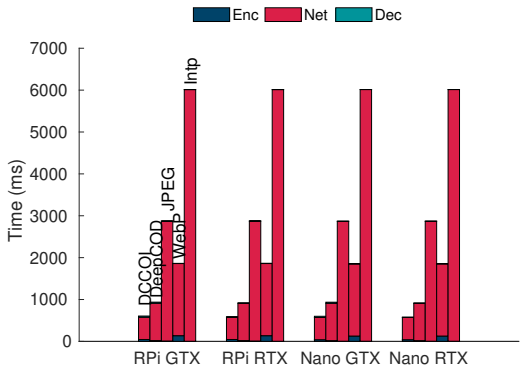

Figure 7: End-to-end offloading latency using LTE

In summary, DCCOI and DeepCOD achieve the lowest end-to-end latency when using LTE and WiFi, respectively. The compression overhead of DCCOI is lower than 50ms on RPi and Nano.

8 CONCLUSION

We present DCCOI, a lightweight, context-aware, and bandwidth-efficient offloading framework for images. DCCOI significantly reduces the offloading bandwidth using a small compression overhead when compared to existing compression techniques with similar accuracy.

REFERENCES

[1] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool. 2017. Soft-to-hard vector quantization for end-to-end learning compressible representations. *arXiv preprint arXiv:1704.00648* (2017).

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

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. 2018. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096* (2018).

[4] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. 2015. Learning complexity-aware cascades for deep pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 3361–3369.

[5] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 155–168.

[6] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. 2020. Server-Driven Video Streaming for Deep Learning Inference. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 557–570.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n. d.]. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[9] Google. 2020. A new image format for the Web. https://developers.google.com/speed/webp

[10] The Independent JPEG Group. 2014. libjpeg. https://github.com/LuaDist/libjpeg

[11] 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.

[12] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu, changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. 2020. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements. https://doi.org/10.5281/zenodo.4154370

[13] N Krishnaraj, Mohamed Elhoseny, M Thenmozhi, Mahmoud M Selim, and K Shankar. 2020. Deep learning model for real-time image compression in Internet of Underwater Things (IoUT). *Journal of Real-Time Image Processing* 17, 6 (2020), 2097–2111.

[14] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[16] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[17] Chrisma Pakha, Aakanksha Chowdhery, and Junchen Jiang. 2018. Reinventing video streaming for distributed vision analytics. In *10th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18)*.

[18] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep face recognition. (2015).

[19] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. 2017. Semantic perceptual image compression using deep convolution networks. In *2017 Data Compression Conference (DCC)*. IEEE, 250–259.

[20] Oren Rippel and Lubomir Bourdev. 2017. Real-Time Adaptive Image Compression. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 2922–2930. http://proceedings.mlr.press/v70/rippel17a.html

[21] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. 2016. Computation offloading and resource allocation for low-power IoT edge devices. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 7–12.

[22] Claude E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

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

[24] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. 2017. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395* (2017).

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

[26] 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.

[27] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. 2019. Self-attention generative adversarial networks. In *International conference on machine learning*. PMLR, 7354–7363.