Compressive Sensing based Asymmetric Semantic Image Compression for Resource-constrained IoT system

Yujun Huang^{1,3†}, Bin Chen^{2,3‡}, Jianghui Zhang^{1,3}, Qiu Han^{4}, Shu-Tao Xia^{1,3}
^{1} Tsinghua Shenzhen International Graduate School, Tsinghua University , ^{2} Harbin Institute of Technology, Shenzhen,
^{3} Peng Cheng Laboratory
^{4} Institute for Network Sciences and Cyberspace, Beijing, National Research Center for Information Science and Technology, Tsinghua University
huangyj20@mails.tsinghua.edu.cn,chenbin2021@hit.edu.cn,jh-zhang21@mails.tsinghua.edu.cn
qiuhan@tsinghua.edu.cn,xiast@sz.tsinghua.edu.cn

ABSTRACT

The widespread application of Internet-of-Things (IoT) and deep learning have made machine-to-machine semantic communication possible. However, it remains challenging to deploy DNN model on IoT devices, due to their limited computing and storage capacity. In this paper, we propose Compressed Sensing based Asymmetric Semantic Image Compression (CS-ASIC) for resource-constrained IoT systems, which consists of a lightweight front encoder and a deep iterative decoder offloaded at the server. We further consider a task-oriented scenario and optimize CS-ASIC for the semantic recognition tasks. The experiment results demonstrate that CS-ASIC achieves considerable data-semantic rate-distortion trade-off, and low encoding complexity over prevailing codecs.

1 INTRODUCTION

Deep learning enables the concept of Artificial Intelligence of Things (AIoT) to become reality, i.e., street pedestrian detection system and unmanned aerial vehicle (UAV) fire-watch system, by analyzing massive data sensed by heterogeneously connected devices. However, due to the limited computing resources, storage space, and battery capacity of IoT devices, deploying DNN models on IoT devices remain an objective obstacle. With mobile edge computing (MEC) [9, 10] and 5G Network [1], a practical solution to the above obstacle is to deploy the DNN models at the powerful edge server near the IoT devices. Therefore, an effective image semantic compression system is required for communication between the IoT devices and edges. Two novel challenges arise in such semantic image compression system: (1) Transmitting overparameterized DNN models and mass of data captured by IoT devices will lead to channel congestion under limited bandwidth; (2) The accuracy of DNN inference will decline due to data recovery error if lossy coding strategy is performed. Therefore, it is crucial to design an asymmetric image compression algorithm for this inference-at-edge semantic image sensing system, that reduces the model size of front encoder and data size for saving bandwidth as well as prevents DNN inference accuracy from degradation.

Next, we discuss some existing image compression methods and their feasibility in inference-at-edge IoT systems. Prevailing lossy image compression algorithms, e.g., JPEG [17], WebP [3], and H.264 [13] perform 2D-DCT, quantization, entropy coding, and intra-frame prediction based on human-vision rate-distortion trade-off, thus taking no account of semantic distortion optimization. Liu et al. [12] proposed a novel heuristic quantization table design to replace the original one in JPEG to retain more high-frequency semantic information for DNN inference. Choi et al. further proposed an improved deep learning method to learn a task-specific JPEG quantization table by adjusting the objective function [6]. But most of their modules, e.g., DCT transform and entropy coding are unchanged and unlearnable restricted by the JPEG framework, thus leading to sub-optimal performance for specific downstream tasks. Besides, deep lossy image compression models [2] achieve considerable rate-distortion performance with a symmetric autoencoder structure. But their heavyweight encoders are not suitable for those front devices with limited computation power and storage resources. To reduce the parameters of DNN model, a popular method is to perform model compression [5]. But the compressed CNN model still contains multiple layers of nonlinear function that may induce high computing complexity for those IoT devices without powerful GPUs. By contrast, compressed sensing (CS) [7] naturally has a lightweight sampling operation. Yuan et al. [19] proposed a gray image compression framework based on CS, but did not consider an efficient extension to color images.

In this paper, we propose Compressed Sensing based Asymmetric Semantic Image Compression model (CS-ASIC) for IoT systems. Our framework simultaneously reduces the computational complexity of the encoder and allocates more computational resources to the decoder to obtain better data recovery and semantic accuracy. Different from previous deep CS works [14, 15] that focus on the trade-off between sample ratio and distortion, our framework optimizes the entire model for rate-distortion performance. Moreover, we propose *split sampling* for more practical IoT systems that splits the sampling process based on different input channels, and reduces the model size for downlink model transmission. Unlike 2D-DCT and random sampling strategy, the sample matrices of *split sampling* are learnable in an end-to-end framework. This work also proposes a deep iterative decoder with *residual fidelity block* (RFB)

† Work was done during an internship at Harbin Institute of Technology, Shenzhen.
‡ Corresponding to: Bin Chen (chenbin2021@hit.edu.cn).

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530520

as its constituent unit for rate-distortion optimization. RFB can improve the image quality by replacing the sparse prior in vanilla CS with learnable prior, correcting recovery error by fidelity step and cumulative error in feature space.

Finally, we adaptively improve the semantic accuracy for specific downstream tasks while balancing the existing rate-distortion performance.

The main contributions can be summarized as follows:

• We design the first deep asymmetric semantic image compression model consisting of a lightweight linear encoder and a deep iterative decoder for the resourced-constrained IoT systems.

• To make model transmission and storage affordable for IoT devices, we propose learnable *split sampling* to reduce model size.

• For low-latent data transmission, we propose a *residual fidelity block* (RFB) based deep iterative decoder, which jointly optimizes the data-semantic rate-distortion performance for downstream tasks.

2 SYSTEM MODEL

Image is an important and common data type in IoT systems to capture information from the physical world. By training semantic image compression models with the image data at cloud/edge servers, the IoT devices can accurately transmit the image stream for human vision and semantic analysis in downstream computer vision tasks, such as pedestrian detection in driver assistance systems, UAV-based fire detection, and flood monitoring.

Fig. 1(a) illustrates the overall workflow of an image semantic compression system, where a deep semantic compression model is deployed in the inference-at-edge IoT system. This system consists of IoT front devices and the cloud/edge server. IoT devices have limited memory and computing capacity, which can only compress the captured data source by a low-complexity encoding strategy before uploading the data stream to the cloud/edge server. The cloud/edge server has powerful computing resources to train DNN model, and perform deep data recovery and semantic analysis. Therefore, for a certain semantic analysis task, i.e, give a well-trained downstream DNN model, our deep asymmetric semantic compression model deployment for such IoT system can be divided into the following four steps:

(1) Model Training: The cloud/edge server trains the overall deep compression model with the captured images.

(2) Model Broadcast: The cloud/edge server broadcasts the trained lightweight linear encoder to IoT devices.

(3) Data Upload: IoT devices encode the captured images to bits streams by the lightweight encoder. The bits streams are then uploaded to the cloud/edge server.

(4) Data Recovery & Semantic Analysis: The cloud/edge server recovers the images with a deep decoder. Semantic information is then extracted by some task-related DNN.

Next, we introduce the major components of our deep asymmetric semantic image compression model as shown in Fig. 1(b). In general, training our model can be separated into two parts: (1) a transmitter network plays as a lightweight encoder, consists of analysis transform, quantization, and entropy encoding. (2) a receiver network consists of a deep decoder and a semantic DNN inference model. The deep decoder also contains similar entropy decoding step and synthesis transformation. Unless specified otherwise, we use the following notations. Denote the input color image as $X \in \mathbb{R}^{H\times W\times 3}$, where H, W denote the height and width of the image, respectively. Then the lossy encoding can be represented as:

$$\hat{\boldsymbol{Y}} = Q(g_a(\boldsymbol{X}; \boldsymbol{\theta}_a)), \tag{1}$$

where $g_a(\cdot)$ is the analysis transform with parameter $\boldsymbol{\theta}_a$ and $Q(\cdot)$ is the quantization operation.

The decoded image can be represented as:

$$\hat{\boldsymbol{X}} = g_s(\hat{\boldsymbol{Y}}; \boldsymbol{\theta}_s), \tag{2}$$

where $g_s(\cdot)$ is the synthesis transform with parameters $\boldsymbol{\theta}_s$.

Furthermore, the extracted semantic information can be obtained from the decoded image, given by:

$$\hat{\boldsymbol{z}} = h(\hat{\boldsymbol{X}}; \boldsymbol{\phi}), \tag{3}$$

where $h(\cdot)$ is a task-related semantic feature extractor.

Then the deep asymmetric semantic image compression model can be trained by the following data-semantic rate-distortion optimization objective:

$$\begin{aligned} &\arg\min_{\boldsymbol{\theta}_a,\boldsymbol{\theta}_s} R + \lambda_1 D_1 + \lambda_2 D_2 \\ &= \arg\min_{\boldsymbol{\theta}_a,\boldsymbol{\theta}_s} \underbrace{\mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}} \left[-\log_2 p_{\hat{\boldsymbol{Y}}} \left(\hat{\boldsymbol{Y}} \right) \right]}_{\text{Rate}} + \lambda_1 \cdot \underbrace{\mathbb{E}_{\boldsymbol{X}\sim p_{\boldsymbol{X}}} \left[d_1 \left(\boldsymbol{X}, \hat{\boldsymbol{X}} \right) \right]}_{\text{Data distortion}} \\ &+ \lambda_2 \cdot \underbrace{\mathbb{E}_{(\boldsymbol{X},\boldsymbol{z})\sim p_{(\boldsymbol{X},\boldsymbol{z})}} \left[d_2 \left(\boldsymbol{z}, \hat{\boldsymbol{z}} \right) \right]}_{\text{Semantic Distortion}}, \end{aligned} \tag{4}$$

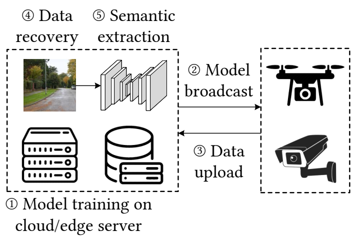
where the first term is the expected bit rate estimated by an entropy model, the second term is the expected human vision based distortion of the reconstructed image, e.g., mean squared error (MSE), and the third term controls the expected semantic distortion $d_2(\cdot,\cdot)$ such as cross entropy with ground truth label $\boldsymbol{z}$, λ_1 and λ_2 are the Lagrange multipliers to control the overall loss.

3 PROPOSED METHOD

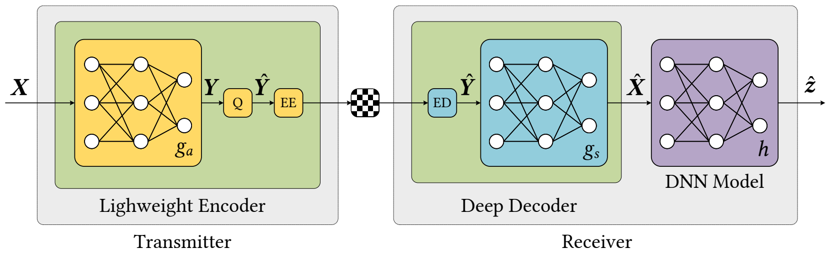
In this section, we describe in detail CS-ASIC, our proposed asymmetric semantic image compression framework for IoT systems.

3.1 Split Sampling based Lightweight Encoder

To better facilitate the resource-constrained IoT devices encoding, we propose Compressed Sensing (CS) based sampling operation to extract the low-dimension image feature due to its considerably low complexity. Given a source image $\boldsymbol{X}$, a common operation in image CS is block-based sampling [8] that divides the image into non-overlapping $B \times B$ blocks: $\{x_i \in R^{3B^2} \mid i = 1, ..., \lceil \frac{H}{B} \rceil \lceil \frac{W}{B} \rceil\}$, and then sampling each block independently. Then the sampling process can be formulated as: $y_i = Ax_i$, where $A \in \mathbb{R}^{M\times 3B^2}$ is a sample matrix such that $M < 3B^2$. If these blocks are sparse with respect to an orthogonal basis Ψ, e.g., discrete cosine transformation (DCT), CS theory proves that if the sample matrix satisfies the *Restricted Isometry Property* (RIP) property, the following sparse



(a) Deep semantic compression system



(b) Deep Asymmetric Semantic Image Compression Model

Figure 1: Proposed framework for inference-at-edge IoT system.

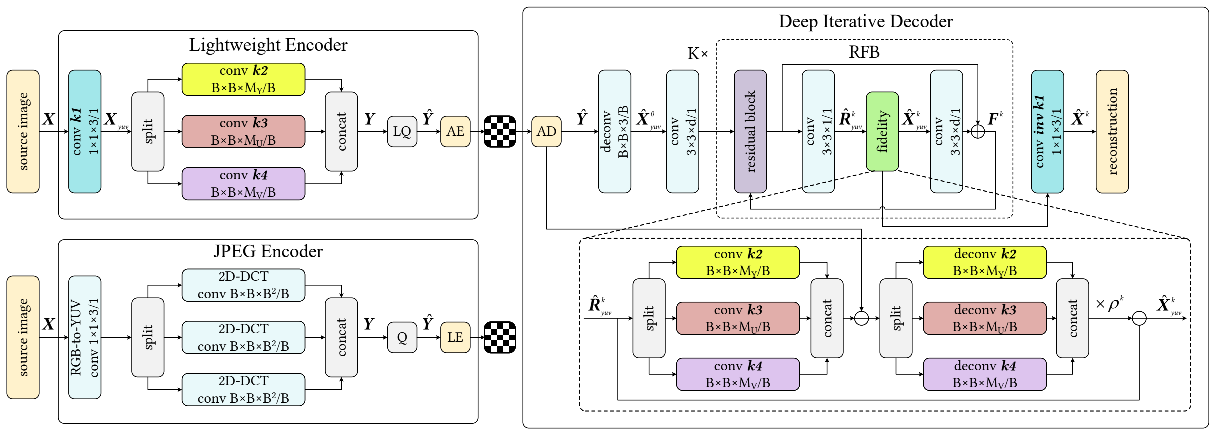


Figure 2: Network architecture of CS-ASIC. The upper left side shows the analysis transform, the lower left side is the encoder of JPEG for comparison, and the right side corresponds to the synthesis transofrm. LQ represents learnable quantization, AE and AD represent arithmetic encoder and arithmetic decoder, respectively, and LE represents lossless encoder. Convolution parameters are denoted as: kernel height×kernel width×number of filters/stride. The RGB-to-YUV transofrm and 2D-DCT of JPEG is described as a special convolution operation for comparison.

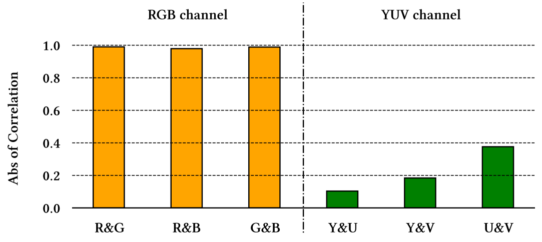


Figure 3: The absolute value of Pearson correlation coefficient within RGB and YUV channel.

solution approximately recover the image block,

$$\arg\min_{x'} \underbrace{\|\Psi x'\|_1}_{\text{Sparse Prior}} + \frac{\rho}{2}\|y_i - Ax'\|_2^2 \tag{5}$$

where ρ is a hyper-parameter. Although the CS sampling operation has linear complexity, the sample matrix size is still large for model transmission. We employ Multilinear Compressive Sensing (MCS) [16] to further reduce the sample matrix size by sampling along each of the dimensions of a given multidimensional input signal by a set of sample matrices. Specifically, sampling the color image signal along one color dimension and two spatial dimensions separately could reduce the size of the sample matrix. Besides, we find that the correlation within the RGB channel is higher than that within the YUV channel as shown in Fig. 3. Therefore, we can sample each YUV channel independently since they have been decorrelated.

Based on the above analysis, we propose a novel two-stage *split sampling*, which first performs RGB-to-YUV transform to decorrelate the color space, then samples each YUV channel separately along spatial dimensions, as shown in Alg 1. Specifically, we perform a block-based CS sampling operation to divide each YUV channel into non-overlapping $B \times B$ blocks: $\{x_{i;j} \in R^{B^2} \mid j \in \{y, u, v\}, \quad i = 1, ..., \lceil \frac{H}{B} \rceil \lceil \frac{W}{B} \rceil\}$, and then sample them with learnable sample matrices. It is worth noting that our block-based sampling step and DCT of JPEG can be both viewed as special types of convolutional operation as shown in Figure 2. Each row of our sample

Algorithm 1 Split Sampling

```
1:  $w_1$, $w_2$: positive learnable scalars;
2:  $W$: RGB-to-YUV transform matrix;
3:  $w_R$, $w_G$, $w_B$: Weights in transform matrix;
4:  $y$, $u$, $v$: Y, U, V channel subscripts;
5:  $split_t(\cdot)$: Split tensor along the $t$-th demension;
6:  $cont_t(\cdot)$: Concatenate tensors along the $t$-th demension;
7:  $S_j(\cdot)$, $j \in \{y, u, v\}$: Convolution with parameters $B \times B \times M_j/B$;
8:  $*_l$: Product along the $l$-th dimension of a tensor with a matrix [11];
9:  $\boldsymbol{Y}$: Sample signal;
10: $w_R, w_G, w_B = \frac{w_1}{w_1+w_2+1}, \frac{w_2}{w_1+w_2+1}, \frac{1}{w_1+w_2+1}$;
11: //RGB-to-YUV transform
12: $\boldsymbol{X}_{yuv} = \boldsymbol{X} *_3 W$,
    where $W = \begin{pmatrix} w_R & \frac{-w_R}{2(1-w_B)} & \frac{1}{2} \\ w_G & \frac{-w_G}{2(1-w_B)} & \frac{-w_G}{2(1-w_R)} \\ w_B & \frac{1}{2} & \frac{-w_B}{2(1-w_R)} \end{pmatrix}$;
13: //Separate channel sampling
14: $(\boldsymbol{X}_y, \boldsymbol{X}_u, \boldsymbol{X}_v) = split_3(\boldsymbol{X}_{yuv})$;
15: $(\boldsymbol{Y}_y, \boldsymbol{Y}_u, \boldsymbol{Y}_v) = (S_y(\boldsymbol{X}_y), S_u(\boldsymbol{X}_u), S_v(\boldsymbol{X}_v))$;
16: $\boldsymbol{Y} = cont_3(\boldsymbol{Y}_y, \boldsymbol{Y}_u, \boldsymbol{Y}_v)$;
17: return $\boldsymbol{Y}$;
```

matrix can be viewed as a filter and the sampling operation is equivalent to a series of convolutional filters. The kernel size and the stride are both $B \times B$. With this viewpoint, we can integrate the sample matrices as learnable parameters into an end-to-end framework as deep learning to learn a lightweight image encoder.

To better illustrate the benefit of our lightweight encoder, we now discuss the model size comparison between our split sampling and the vanilla block-based sampling operation (**??**). The total size of the sample matrices of split sampling is $B^2(M_y + M_u + M_v)$, while the size of vanilla one is $3B^2M$. Clearly, our split sampling reduces the size by 3 times if $M_y + M_u + M_v = M$, thus contributing to a more efficient lightweight encoder. Besides, as shown in Fig. 2 and subsequent experimental results, our learnable encoder and JPEG encoder have similar linear structures and comparable complexity.

3.2 Residual Fidelity Block based Deep Iterative Decoder

Although we can realize low-latency model transmission in the downlink by the proposed *split sampling* operation, the amount of data can be huge in the uplink, which causes high latency for data transmission. Therefore, we replace the sparse prior in vanilla CS optimization (5) with learnable prior to improve rate-distortion performance. Specifically, we employ a deep reconstruction function $f_{\theta}(\cdot)$ to learn a mapping from the quantized sample signal $\hat{\boldsymbol{Y}}$ to the recovered YUV image $\hat{\boldsymbol{X}}_{yuv}$, and formulate the objective below.

$$\begin{aligned} \arg\min\nolimits_{\theta} \quad & \underbrace{\|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_2^2}_{\text{Learnable Prior through } f_{\theta}(\cdot)} + \frac{\rho}{2} \sum_{j \in \{y,u,v\}} \|\hat{\boldsymbol{Y}}_j - S_j(\hat{\boldsymbol{X}}_j)\|_2^2 \\ \text{s.t.} \quad & \hat{\boldsymbol{X}}_{yuv} = f_{\theta}(\hat{\boldsymbol{Y}}) \ \text{ and } \ \hat{\boldsymbol{X}} = \hat{\boldsymbol{X}}_{yuv} *_3 W^{-1}. \end{aligned} \tag{6}$$

With (6), we can learn the data prior from the learnable network parameters θ, that greatly improves the reconstruction efficiency. Moreover, We adopt the iterative gradient unrolling method [4] to solve the problem (6). Firstly, we denote the initial reconstruction

Algorithm 2 Residual fidelity block based decoding

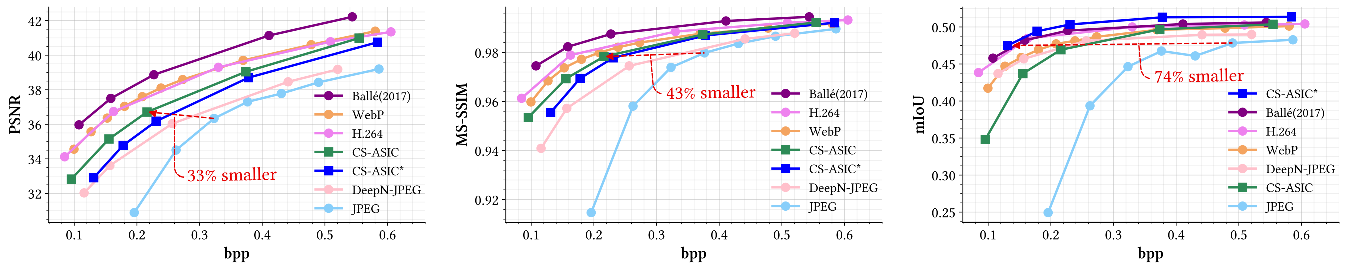
```
1:  $k$: The $k$-th iteration;
2:  $Rb^k(\cdot)$: Residual block;
3:  $C(\cdot)$: Convolution with parameters $3 \times 3 \times d/1$;
4:  $C_1^k(\cdot)$: Convolution with parameters $3 \times 3 \times 1/1$;
5:  $C_2^k(\cdot)$: Convolution with parameters $3 \times 3 \times d/1$;
6:  $D(\cdot)$: Deconvolution with parameters $B \times B \times (M_y + M_u + M_v)/B$;
7:  $S_j^T(\cdot)$, $j \in \{y, u, v\}$: Deconvolution with parameters $B \times B \times M_j/B$
    and the same kernel as $S_j(\cdot)$;
8:  $\boldsymbol{F}^k$: Output feature;
9:  $\hat{\boldsymbol{R}}^k_{yuv}$: Intermediate deep YUV recovery;
10: $\hat{\boldsymbol{X}}^k_{yuv}$: Last fidelity YUV recovery;
11: Other symbols are decribed in Alg. 1;
12: $\hat{\boldsymbol{X}}^0_{yuv} = D(\hat{\boldsymbol{Y}})$;
13: $\boldsymbol{F}^0 = C(\hat{\boldsymbol{X}}^0_{yuv})$;
14: $\hat{\boldsymbol{X}}^0 = \hat{\boldsymbol{X}}^0_{yuv} *_3 W^{-1}$;
15: //K iterations of RFB
16: for $k$ in $[1 \,..\, K]$ do
17:     //Deep Recovery
18:     $\hat{\boldsymbol{R}}^k_{yuv} = \hat{\boldsymbol{X}}^{k-1}_{yuv} + C_1^k(Rb^k(\boldsymbol{F}^{k-1}))$;
19:     //Fidelity
20:     $(\hat{\boldsymbol{R}}^k_y, \hat{\boldsymbol{R}}^k_u, \hat{\boldsymbol{R}}^k_v) = split_3(\hat{\boldsymbol{R}}^k_{yuv})$;
21:     $(\hat{\boldsymbol{Y}}^k_y, \hat{\boldsymbol{Y}}^k_u, \hat{\boldsymbol{Y}}^k_v) = (S_y(\hat{\boldsymbol{R}}^k_y), S_u(\hat{\boldsymbol{R}}^k_u), S_v(\hat{\boldsymbol{R}}^k_v))$;
22:     $\hat{\boldsymbol{Y}}^k = cont_3(\hat{\boldsymbol{Y}}^k_y, \hat{\boldsymbol{Y}}^k_y, \hat{\boldsymbol{Y}}^k_y)$;
23:     $(\hat{\boldsymbol{Y}}^k_y - \hat{\boldsymbol{Y}}_y, \hat{\boldsymbol{Y}}^k_u - \hat{\boldsymbol{Y}}_u, \hat{\boldsymbol{Y}}^k_v - \hat{\boldsymbol{Y}}_v) = split_3(\hat{\boldsymbol{Y}}^k - \hat{\boldsymbol{Y}})$;
24:     $\hat{\boldsymbol{X}}^k_{yuv} = \hat{\boldsymbol{R}}^k_{yuv} - \rho^k cont_3(S_y^T(\hat{\boldsymbol{Y}}^k_y - \hat{\boldsymbol{Y}}_y), S_u^T(\hat{\boldsymbol{Y}}^k_u - \hat{\boldsymbol{Y}}_u), S_v^T(\hat{\boldsymbol{Y}}^k_v -$
        $\hat{\boldsymbol{Y}}_v))$;
25:     $\hat{\boldsymbol{X}}^k = \hat{\boldsymbol{X}}^k_{yuv} *_3 W^{-1}$;
26:     //Fidelity Skip Connection
27:     $\boldsymbol{F}^k = Rb^k(\boldsymbol{F}^{k-1}) + C_2^k(\hat{\boldsymbol{X}}^k_{yuv})$;
28: end for
29: return $[\hat{\boldsymbol{X}}^0 \,..\, \hat{\boldsymbol{X}}^K]$;
```

$\hat{\boldsymbol{X}}^0_{yuv} = f^0_{\theta^0}(\hat{\boldsymbol{Y}})$. Then we can obtain the *residual fidelity block (RBF)* based iterative decoding as follows:

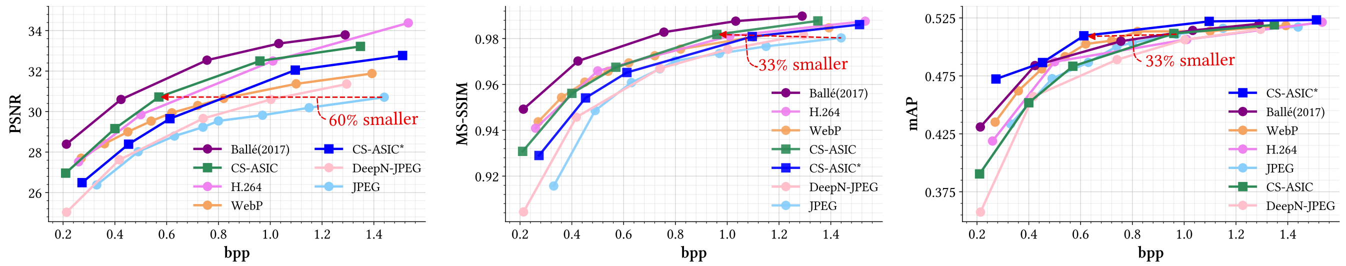
$$\begin{aligned} \arg\min_{\theta^1,\ldots,\theta^K} \quad & \sum_{k=0}^{K} \|\boldsymbol{X} - \hat{\boldsymbol{X}}^k\|_2^2 \\ \text{s.t.} \quad & \underbrace{\hat{\boldsymbol{R}}^k_{yuv} = f^k_{\theta^k}\left(\hat{\boldsymbol{X}}^{k-1}_{yuv}\right)}_{\text{Step 1}}, \quad \underbrace{\hat{\boldsymbol{X}}^k_j = \hat{\boldsymbol{R}}^k_j - \rho^k \boldsymbol{G}^k_j}_{\text{Step 2}}, \\ & \boldsymbol{G}^k_j = S_j^T(S_j(\hat{\boldsymbol{R}}^k_j) - \hat{\boldsymbol{Y}}_j), \ k \in \{1, 2, \cdots, K\}, \ j \in \{y, u, v\} \end{aligned} \tag{7}$$

where K is the number of total iterations, $\hat{\boldsymbol{R}}^k_j$ is the intermediate recovery, while $\hat{\boldsymbol{X}}^k_j$ ($j \in \{y, u, v\}$) is the final recovery of the j channel in the k-th iteration, $\boldsymbol{G}^k$ denotes the gradients of $\frac{1}{2}\|\hat{\boldsymbol{Y}}_j - S_j(\hat{\boldsymbol{X}}_j)\|_2^2$ (the second term of Eq. (6)) at $\hat{\boldsymbol{X}}_j = \hat{\boldsymbol{R}}^k_j$, and S_j^T is the deconvolution operation [20] to perform matrix multiplication between the transpose of the sample matrix and a tensor.

Different from (6), we regard step 1 and step 2 of (7) as one iteration of decoding. We call step 2 the *fidelity step* as it can help to correct the deep reconstruction error generated by step 1. Based on



(a) Cityscapes



(b) KITTI

Figure 4: The data-semantic rate-distortion performance comparison of JPEG, Webp, H.264, DeepN-JPEG, Balle(2017) and CS-ASIC on Cityscapes and KITTI datasets.

the iterative optimization strategy (7), we can view it as an iterative deep recovery network with residual fidelity block. As shown in Alg. 2, the RFB includes three phases: i) Deep Recovery; ii) Fidelity; iii) Fidelity Skip Connection. Phase 1 is corresponding to step 1, where the residual block is used to learn image prior and helps to better recover the last outputs. Phase 2 is a separate channel gradient updating process corresponding to step 2. In phase 3, the feature extracted from fidelity recovery is added to the original feature to correct the cumulative error. With this iterative decoding, we can achieve low data-semantic distortion by the overall objective of (4).

4 EXPERIMENTS

In this section, we evaluate CS-ASIC with other prevailing codecs in resource-constrained IoT systems with the following three objectives: 1) comparing CS-ASIC with other image compression methods and demonstrating that our method maintains a good balance among compression rate, encoding complexity, human vision based distortion and semantic accuracy; 2) showing the efficiency and extensibility of CS-ASIC in multi-task semantic recognition scenario; 3) testing the deployment cost of CS-ASIC and other methods.

4.1 Experiment Setup

Experimental Platform. We evaluate CS-ASIC and compare it with JPEG, WebP, H.264, DeepN-JPEG, Ballé (2017) on Jetson Nano b01 to simulate resource-constrained IoT devices. The Jetson Nano b01 contains a 4 core ARM A57 CPU and a 128 core Maxwell GPU.
Model Setting. We set the width and height of an image block B as 10. The number of measurements for YUV channels M_y, M_u and M_v is 28, 10, and 10, respectively. The number of iteration K is set to 2. The dimension of feature maps d is 64. The activation function in the residual block is Leaky ReLU. In the training phase, 30 images are randomly sampled from a dataset and cropped to 96×96 sub-images as a mini-batch. The learning rate is 1e-4. The total number of training iteration is 3×10^5.
Evaluation Dataset. We conduct our experiments on datasets Cityscapes and KITTI. The Cityscapes dataset is a large-scale dataset with high-quality pixel-level annotations of 5000 street scenes images from 50 different cities. It contains 19 foreground objects for image segmentation. KITTI is a primary dataset for image processing technologies in the field of autonomous driving. We use the Object Detection Evaluation 2012 subset for object detection and KITTI semantic segmentation benchmark for image segmentation.
Metrics. Bits per pixel (bpp) is used to measure the compression ratio. Peak signal-to-noise ratio (PSNR) and multi-scale structural similarity (MS-SSIM) [18] are used to measure image quality. Mean intersection over union (mIoU) and mean average precision (mAP) are used to measure the accuracy of image segmentation and object detection, respectively.

4.2 Evaluation Results and Discussion

Data-semantic Rate-distortion Performance. Fig. 4 compares CS-ASIC with JPEG, WebP, H.264, DeepN-JPEG, Ballé (2017) versus different bpps on Cityscapes and KITTI dataset, in terms of PSNR, MS-SSIM, mIoU, and mAP. CS-ASIC is trained by conventional data rate-distortion loss (i.e, first two terms of (4)) and CS-ASIC* is trained by (4). We observe that WebP and H.264 are better than JPEG. This is because they have intra-prediction to decorrelate the neighboring blocks. DeepN-JPEG is better than JPEG on image segmentation task, but worse on object detection task. The performance of Ballé (2017) is better than CS-ASIC on data rate-distortion performance, but requires high complexity for encoding. The compression rate of CS-ASIC* is 1.5~3.8 times compared to JPEG with similar inference accuracy. The compression rate of CS-ASIC is 1.5~2.5 times compared to JPEG with similar data distortion.

Multi-task Scenario. To better demonstrate the efficiency and extensibility of CS-ASIC, we further train it (denoted as CS-ASIC**) by (4) with semantic distortion deriving from both object detection and image segmentation of KITTI dataset. As shown in Fig. 5, CS-ASIC** achieves the best inference accuracy on object detection and image segmentation task compared with other codecs.

Deployment Cost on IoT Devices. Table 1 illustrate the floating point operations (FLOPs), runtime, and the size of the encoder of CS-ASIC, JPEG, WebP, H.264 and Ballé (2017), where FLOPs is the number of floating-point operations for encoding an 80×80 image and the runtime is tested on Jetson Nano b01. We observe that our proposed CS-ASIC and JPEG have comparable low complexity. By contrast, WebP and H.264 cost more resources due to their high-complexity intra-frame predictions. We also observe that deep symmetric image compression like Ballé (2017) costs about 170 times more than CS-ASIC on CPU implementation and 35 times more on GPU implementation because their encoder consists of high-dimensional CNN layers.

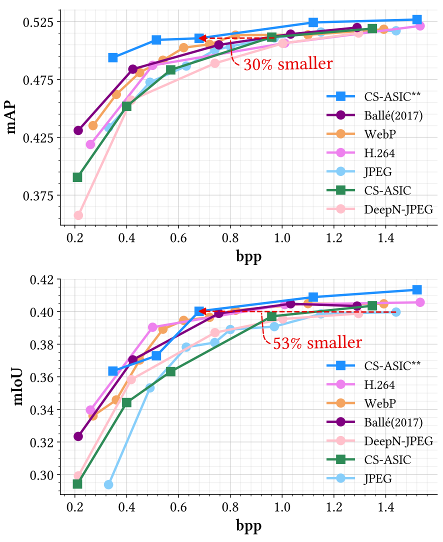


Figure 5: The semantic rate-distortion performance on the object detection and image segmentation of KITTI dataset.

5 CONCLUSION

In this paper, we propose the first asymmetric semantic image compression model for the inference-at-edge IoT systems, called CS-ASIC. By deploying a lightweight learnable encoder at the front encoder and a deep iterative reconstruction network at the central decoder, we validate that the proposed CS-ASIC outperforms other methods under low-complexity encoding constraint. We plan to extend the CSDIC framework to heterogeneous front devices to realize adaptive encoding and design a novel video coding algorithm suitable for this resource-constrained scenario.

Table 1: FLOPs, runtime and number of parameters of the encoder of CS-ASIC, JPEG, WebP, H.264 and Ballé(2017).

Encoder	FLOPs	Runtime		Model size (# params)
		CPU	GPU	
CS-ASIC	0.37M	0.060s	0.049s	5.10k
JPEG	0.38M	0.062s	-	4.80k
WebP	1.00M+	0.43s	-	9.61k+
H.264	1.65M+	0.67s	-	9.61k+
Ballé(2017)	584.7M	10.45s	1.71s	2.89M

6 ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China under grant 62171248, the PCNL KEY project (PCL2021A07), and the Guangdong Basic and Applied Basic Research Foundation under grant 2021A1515110066.

REFERENCES

[1] M. Agiwal, A. Roy, and N. Saxena. 2016. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 18, 3 (2016), 1617–1655.
[2] J. Ballé, V. Laparra, and E. P. Simoncelli. 2017. End-to-end optimized image compression. In *Int'l Conf on Learning Representations (ICLR)*. Toulon, France. https://arxiv.org/abs/1611.01704 Available at http://arxiv.org/abs/1611.01704.
[3] Somnath Banerjee and Vikas Arora. 2011. Webp compression study. *code. google. com/speed/webp/docs/webp study.html* (2011).
[4] A. Beck and M. Teboulle. 2009. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *Siam Journal on Imaging Sciences* (2009), 183–202.
[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).
[6] J. Choi and B. Han. 2020. Task-Aware Quantization Network for JPEG Image Compression. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 309–324.
[7] Y. C. Eldar and G. Kutyniok. 2012. *Compressed Sensing: Theory and Applications.* Cambridge University Press.
[8] L. Gan. 2007. Block compressed sensing of natural images. In *2007 15th International conference on digital signal processing*. IEEE, 403–406.
[9] Y Gao, W Hu, and K Ha. 2015. Are cloudlets necessary? *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139* (2015).
[10] W. Hu, Y. Gao, K. Ha, J. Wang, and M. Satyanarayanan. 2016. Quantifying the Impact of Edge Computing on Mobile Applications. In *Acm Sigops Asia-pacific Workshop on Systems*.
[11] T. G. Kolda and B. W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (aug 2009), 455–500.
[12] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan. 2018. DeepN-JPEG: A Deep Neural Network Favorable JPEG-based Image Compression Framework. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.
[13] I. E. Richardson. 2004. H. 264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. *John Wiley & Sons* (2004).
[14] W. Shi, F. Jiang, S. Liu, and D. Zhao. 2019. Scalable convolutional neural network for image compressed sensing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12290–12299.
[15] W. Shi, F. Jiang, S. Liu, and D. Zhao. 2020. Image Compressed Sensing Using Convolutional Neural Network. *IEEE Transactions on Image Processing* 29 (2020), 375–388.
[16] D. T. Tran, M. Yamac, A. Degerli, M. Gabbouj, and A. Iosifidis. 2019. Multilinear Compressive Learning. *CoRR* abs/1905.07481 (2019). arXiv:1905.07481
[17] G. K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38, 1 (1992), xviii–xxxiv.
[18] Z. Wang, E. P. Simoncelli, and A. C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. Ieee, 1398–1402.
[19] X. Yuan and R. Haimi-Cohen. 2020. Image Compression Based on Compressive Sensing: End-to-End Comparison With JPEG. *IEEE Transactions on Multimedia* 22, 11 (2020), 2889–2904.
[20] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. 2010. Deconvolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2528–2535.