1. Image Compression Codecs

As image compression baselines, we use JPEG, WebP [9], and BPG [6]. For JPEG and WebP, we follow the implementations in Pillow [1] and investigate the rate-distortion (RD) tradeoff for the combination of the codec and pretrained downstream models by tuning the quality parameter in range of 10 to 100. Since BPG is not available in Pillow, our implementation follows [6] and we tune the quality parameter in range of 0 to 50 to observe the RD curve. We use the x265 encoder with 4:4:4 subsampling mode and 8-bit depth for YCbCr color space, following [5].

2. Quantization

This section briefly introduces the quantization technique used in both proposed methods and neural baselines with entropy coding.

2.1. Encoder and Decoder Optimization

As entropy coding requires discrete symbols, we leverage the method that is firstly proposed in [3] to learn a discrete latent variable. During the training stage, the quantization is simulated with a uniform noise to enable gradient-based optimization:

\[ z = f_\theta(x) + U\left( -\frac{1}{2}, \frac{1}{2} \right). \]  

(S1)

During the inference session, we round the encoder output to the nearest integer for entropy coding and the input of the decoder:

\[ z = \lceil f_\theta(x) \rceil. \]  

(S2)

2.2. Prior Optimization

For entropy coding, a prior that can precisely fit the distribution of the latent variable reduces the bitrate. However, the prior distributions such as Gaussian and Logistic distributions are continuous, which is not directly compatible with discrete latent variables. Instead, we use the cumulative of a continuous distribution to approximate the probability mass of a discrete distribution. [3]:

\[ P(z) = \int_{z-rac{1}{2}}^{z+rac{1}{2}} p(t)dt, \]  

(S3)

where \( p \) is the prior distribution we choose, and \( P(z) \) is the corresponding probability mass under the discrete distribution \( P \). The integral can easily be computed with the Cumulative Distribution Function (CDF) of the continuous distribution.

3. Neural Image Compression

In this section, we describe the experimental setup that we used for the neural image compression baselines.

3.1. Network Architecture

Factorized prior model [4]. This model consists of 4 convolutional layers for encoding and 4 deconvolutional layers for decoding. Each layer follows (128, 5, 2, 2) configuration in the format (number of channels, kernel size, stride, padding). We also use the simplified version of generalized divisive normalization (GDN) and inversed GDN (IGDN) [2] as activation functions for the encoder and decoder, respectively. The prior distribution uses a univariate non-parametric density model, whose cumulative distribution is parameterized by a neural network [4].

Mean-scale hyperprior model. We use exactly the same architecture described in [21].

3.2. Training

All the models are trained on a high-resolution dataset with around 2,700 images collected from DIV2K dataset [1] and CLIC dataset [27]. During training, we apply random crop size (256, 256) to the images and set the batch size as 8. We also use Adam [14] optimizer with \( 10^{-4} \) learning rate to train the model for 900,000 steps, and then the learning rate is decayed to \( 10^{-5} \) for another 100,000 steps.
4. Channel Reduction and Bottleneck Quantization

A combination of channel reduction and bottleneck quantization (CR + BQ) is a popular approach in studies on split computing [8][19][25][20], and we refer to the approach as a baseline.

4.1. Network Architecture

Image classification. We reuse the architectures of encoder and decoder from Matsubara et al. [19] introduced in ResNet [11] and validated on the ImageNet (ILSVRC 2012) dataset [23]. Following the study, we explore the rate-distortion (RD) tradeoff by varying the number of channels in a convolution layer (2, 3, 6, 9, and 12 channels) placed at the end of the encoder and apply a quantization technique (32-bit floating point to 8-bit integer) [13] to the bottleneck after the training session.

Object detection and semantic segmentation. Similarly, we reuse the encoder-decoder architecture used as ResNet-based backbone in Faster R-CNN [22] and Mask R-CNN [10] for split computing [20]. The same ResNet-based backbone is used for RetinaNet [15] and DeepLabv3 [7]. Again, we examine the RD tradeoff by controlling the number of channels in a bottleneck layer (1, 2, 3, 6, and 9 channels) and apply the same post-training quantization technique [13] to the bottleneck.

4.2. Training

Using ResNet-50 [11] pretrained on the ImageNet dataset as a teacher model, we train the encoder-decoder introduced to a copy of the teacher model, that is treated as a student model for image classification. We apply the generalized head network distillation (GHND) [20] to the introduced encoder-decoder in the student model. The model is trained on the ImageNet dataset to mimic the intermediate features from the last three residual blocks in the teacher (ResNet-50) by minimizing the sum of squared error losses. Using the Adam optimizer [14], we train the student model on the ImageNet dataset for 20 epochs with the training batch size of 32. The initial learning rate is set to $10^{-3}$ and reduced by a factor of 10 at the end of the 5th, 10th, and 15th epochs.

Similarly, we use ResNet-50 models in RetinaNet with FPN and DeepLabv3 pretrained on COCO 2017 dataset [16] as teachers, and apply the GHND to the students for the same dataset. The training objective, the initial learning rate, and the number of training epochs are the same as those for the classification task. We set the training batch size to 2 and 8 for object detection and semantic segmentation tasks, respectively. The learning rate is reduced by a factor of 10 at the end of the 5th and 15th epochs.

5. Proposed Student Model

This section presents the details of student models and training methods we propose in this study.

5.1. Network Architecture

As illustrated in Fig. [S1], our encoder $f_o$ is composed of convolution and GDN [2] layers followed by a quantizer described in Section [2]. Similarly, our decoder $g_o$ is designed with convolution and inversed GDN (IGDN) layers to have the output tensor shape match that of the first residual block in ResNet-50 [11]. For image classification, the entire architecture of our entropic student model consists of the encoder and decoder followed by the last three residual blocks, average pooling, and fully-connected layers in ResNet-50. For object detection and semantic segmentation, we replace ResNet-50 (used as a backbone) in RetinaNet [15] and DeepLabv3 [7] with our student model for image classification.

5.2. Two-stage Training

Here, we describe the two-stage method we proposed to train the entropic student models.

Image classification. Using the ImageNet dataset, we put our focus on the introduced encoder and decoder at the first stage of training and then freeze the encoder to fine-tune all the subsequent layers at the second stage for the target task. At the 1st stage, we train the student model for 10 epochs to mimic the behavior of the first residual block in the teacher model (pretrained ResNet-50) in a similar way to [20] but with the rate term to learn a prior for entropy coding. We use Adam optimizer with batch size of 64 and an initial learning rate of $10^{-3}$. The learning rate is decreased by a factor of 10 after the end of the 5th and 8th epochs.

Once we finish the 1st stage, we fix the parameters of the encoder that has learnt compressed features at the 1st stage and fine-tune all the other modules, including the decoder for the target task. By freezing the encoder’s parameters, we can reuse the encoder for different tasks. The rest of the layers can be optimized to adopt the compressible features for the target task. Note that once the encoder is frozen, we also no longer optimize both the prior and encoder, which means we can directly use rounding to quantize the latent variable. With the encoder frozen, we apply a standard knowledge distillation technique [12] to achieve better model accuracy, and the concrete training objective is formulated as follows:

$$
\mathcal{L} = \alpha \cdot \mathcal{L}_{ch}(\hat{y}, y) + (1 - \alpha) \cdot \tau^2 \cdot \mathcal{L}_{KL}(\mathbf{o}^S, \mathbf{o}^T), \quad (S4)
$$

where $\mathcal{L}_{ch}$ is a standard cross entropy, $\hat{y}$ indicates the model’s estimated class probabilities, and $y$ is the annotated object category. $\alpha$ and $\tau$ are both hyperparameters, and $\mathcal{L}_{KL}$
We reuse the entropic student model trained on the ImageNet dataset in place of ResNet-50 in RetinaNet [15] and DeepLabv3 [7] (teacher models). Note that we freeze the parameters of the encoder trained on the ImageNet dataset to make the encoder sharable for multiple tasks. Reusing the encoder trained on the ImageNet dataset is a reasonable approach as 1) the ImageNet dataset contains a larger number of training samples (approximately 10 times more) than those in the COCO 2017 dataset [16]; 2) models using an image classifier as their backbone frequently reuse model weights trained on the ImageNet dataset [22, 15].

To adapt the encoder for object detection, we train the decoder for 3 epochs at the 1st stage in the same way we train those for image classification (but with the encoder frozen). The optimizer is Adam [14], and the training batch size is 6. The initial learning rate is set to $10^{-3}$ and reduced to $10^{-4}$ after the first 2 epochs. At the 2nd stage, we fine-tune the whole model except its encoder for 2 epochs by the SGD optimizer with learning rates of $10^{-3}$ and $10^{-4}$ for the 1st and 2nd epochs, respectively. We set the training batch size to 6 and follow the training objective in [15], which is a combination of L1 loss for bounding box regression and Focal loss for object classification.

### Semantic segmentation.
For semantic segmentation, we train DeepLabv3 in a similar way. At the 1st stage, we freeze the encoder and train the decoder for 5 epochs, using Adam optimizer with batch size of 8. The initial learning rate is $10^{-3}$ and decreased to $10^{-4}$ after the first 3 epochs. At the 2nd stage, we train the entire model except for its encoder for 6 epochs. We minimize a standard cross entropy loss, using the SGD optimizer. The initial learning rates for the body and the sub-branch (auxiliary module) are $2.5 \times 10^{-3}$ and $2.5 \times 10^{-2}$, respectively. Following [7], we reduce the learning rate after each iteration as follows:

$$lr = lr_0 \times \left(1 - \frac{N_{\text{iter}}}{N_{\text{max_iter}}} \right)^{0.9}, \quad (S6)$$

where $lr_0$ is the initial learning rate. $N_{\text{iter}}$ and $N_{\text{max_iter}}$ indicate the accumulated number of iterations and the total number of iterations, respectively.

### 5.3. End-to-end Training

In this work, the end-to-end training approach for feature compression [26] is treated as a baseline and applied to our entropic student model without teacher models.

### Image classification.
Following the end-to-end training approach [26], we train our entropic student model from

---

**Figure S1:** Our encoder and decoder introduced to ResNet-50. k: kernel size, s: stride, p: padding.

...
scratch. Specifically, we use Adam\[^{14}\] optimizer and cosine decay learning rate schedule\[^{17}\] with an initial learning rate of $10^{-3}$ and weight decay of $10^{-4}$. Based on their training objectives (Eq. S7), we train the model for 60 epochs with batch size of 256.\[^{3}\] Note that Singh \textit{et al.}\[^{26}\] evaluate the accuracy of their models on a $299 \times 299$ center crop. Since the pretrained ResNet-50 expects the crop size of $224 \times 224$, we use the crop size for all the considered classifiers to highlight the effectiveness of our approach.

\[
\mathcal{L} = \mathcal{L}_{\text{cls}}(\hat{y}, y) - \beta \log p_{\phi}(f_{\theta}(x) + \epsilon), \quad \epsilon \sim \text{Unif}(-\frac{1}{2}, \frac{1}{2})
\]  
(S7)

**Object detection.** Reusing the model trained on the ImageNet dataset with the end-to-end training method, we fine-tune RetinaNet\[^{15}\]. Since we empirically find that a standard transfer learning approach\[^{5}\] to RetinaNet with the model trained by the baseline method did not converge, we apply the 2nd stage of our fine-tuning method described above to the RetinaNet model. The hyperparameters are the same as above, but the number of epochs for the 2nd stage training is 5.

**Semantic segmentation.** We fine-tune DeepLabv3\[^{7}\] with the same model trained on the ImageNet dataset. Using the SGD optimizer with an initial learning rate of 0.01, momentum of 0.9, and weight decay of 0.001, we minimize a standard cross entropy loss. The learning rate is adjusted by Eq. S6 and we train the model for 30 epochs with batch size of 16.

### 6. End-to-End Prediction Latency

In this section, we provide the detail of the end-to-end prediction latency evaluation shown in this work. Figures S2 and S3 show the breakdown of the end-to-end latency per image for Raspberry Pi 4 (RPI4) and NVIDIA Jetson TX2 (JTX2) as mobile devices, respectively. For each of the configurations we considered, we present 1) local processing delay (encoding delay on mobile device), 2) communication delay to transfer the encoded (compressed) data to edge server by LoRa\[^{24}\], and 3) server processing delay to decode the data transferred from mobile device.

---

\[^{3}\]For the ImageNet dataset, Singh \textit{et al.}\ train their models for 300k steps with batch size of 256 for 1.28M training samples, which is equivalent to 60 epochs ($= \frac{300 \times 256}{1.28 \times 10^6}$).

\[^{5}\]\url{https://pytorch.org/vision/stable/models.html#classification}

\[^{6}\]\url{https://github.com/pytorch/vision/tree/master/references/detection}
and complete the inference pipeline on edge server (ES). Following [18][19][20], we compute the communication delay by dividing transferred data size by the available data rate, 37.5 Kbps (LoRa [24]) in this paper. For all the considered approaches, we use the data points with about 74% accuracy in our experiments with the ImageNet dataset.

From the figures, we can confirm that the communication delay is dominant in the end-to-end latency for all the approaches we considered, and the third component (server processing delay) is also negligible as the edge server has more computing power that the mobile devices have. Overall, our entropic student model successfully saves the end-to-end prediction latency by compressing the data to be transferred to edge server with a small portion of computing cost on mobile device.

References


