Supervised Compression for Resource-Constrained Edge Computing Systems - Supplementary Material -

Yoshitomo Matsubara Ruihan Yang Marco Levorato Stephan Mandt
Department of Computer Science, University of California, Irvine
{yoshitom, ruihan.yang, levorato, mandt}@uci.edu

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:

$$\mathbf{z} = f_\theta(\mathbf{x}) + \mathcal{U}(-\frac{1}{2}, \frac{1}{2}). \quad \text{(S1)}$$

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

$$\mathbf{z} = \lfloor f_\theta(\mathbf{x}) \rceil. \quad \text{(S2)}$$

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]:

$$P(\mathbf{z}) = \int_{\mathbf{z}-\frac{1}{2}}^{\mathbf{z}+\frac{1}{2}} p(t)\mathrm{d}t, \quad \text{(S3)}$$

where p is the prior distribution we choose, and $P(\mathbf{z})$ 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.

[1] https://python-pillow.org/

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_θ is composed of convolution and GDN [2] layers followed by a quantizer described in Section 2. Similarly, our decoder g_ϕ 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:

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

where $\mathcal{L}_{\text{cls}}$ is a standard cross entropy. $\hat{\mathbf{y}}$ indicates the model's estimated class probabilities, and $\mathbf{y}$ is the annotated object category. α and τ are both hyperparameters, and $\mathcal{L}_{\text{KL}}$

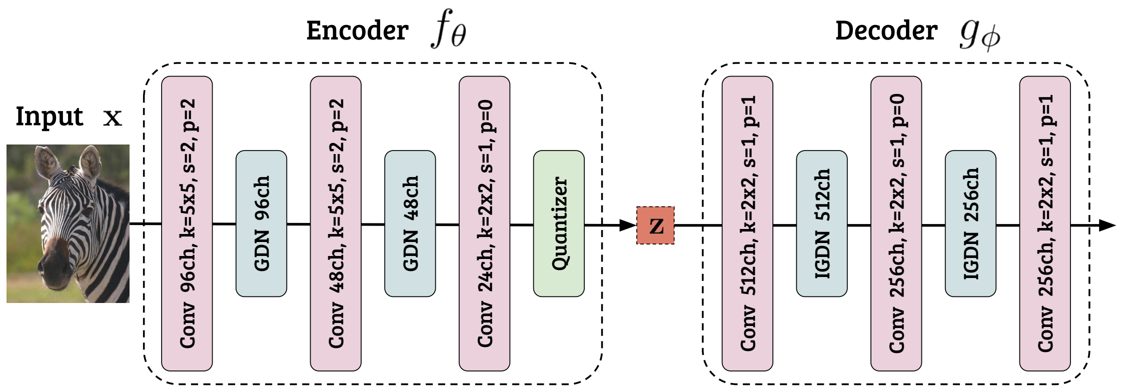
Encoder f_θ
Decoder g_ϕ
Input x
Conv 96ch, k=5x5, s=2, p=2
GDN 96ch
Conv 48ch, k=5x5, s=2, p=2
GDN 48ch
Conv 24ch, k=2x2, s=1, p=0
Quantizer
z
Conv 512ch, k=2x2, s=1, p=1
IGDN 512ch
Conv 256ch, k=2x2, s=1, p=0
IGDN 256ch
Conv 256ch, k=2x2, s=1, p=1

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

is the Kullback-Leibler divergence. $\mathbf{o}^{\mathrm{S}}$ and $\mathbf{o}^{\mathrm{T}}$ represent the *softened* output distributions from student and teacher models, respectively. Specifically, $\mathbf{o}^{\mathrm{S}} = [o_1^{\mathrm{S}}, o_2^{\mathrm{S}}, \dots, o_{|\mathcal{C}|}^{\mathrm{S}}]$ where $\mathcal{C}$ is a set of object categories considered in target task. o_i^{S} indicates the student model's softened output value (scalar) for the i-th object category:

$$o_i^{\mathrm{S}} = \frac{\exp\left(\frac{v_i}{\tau}\right)}{\sum_{k \in \mathcal{C}} \exp\left(\frac{v_k}{\tau}\right)}, \tag{S5}$$

where τ is a hyperparameter defined in Eq. S4 and called *temperature*. v_i denotes a logit value for the i-th object category. The same rules are applied to $\mathbf{o}^{\mathrm{T}}$ for teacher model.

For the 2nd stage, we use the stochastic gradient descent (SGD) optimizer with an initial learning rate of 10^{-3}, momentum of 0.9, and weight decay of 5×10^{-4}. We reduce the learning rate by a factor of 10 after the end of the 5th epoch, and the training batch size is set to 128. The balancing weight α and temperature τ for knowledge distillation are set to 0.5 and 1, respectively.

Object detection. 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 5 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)[2] are 2.5×10^{-3} and 2.5×10^{-2}, respectively. Following [7], we reduce the learning rate after each iteration as follows:

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

where lr_0 is the initial learning rate. N_{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

[2] `https://github.com/pytorch/vision/tree/master/references/segmentation`

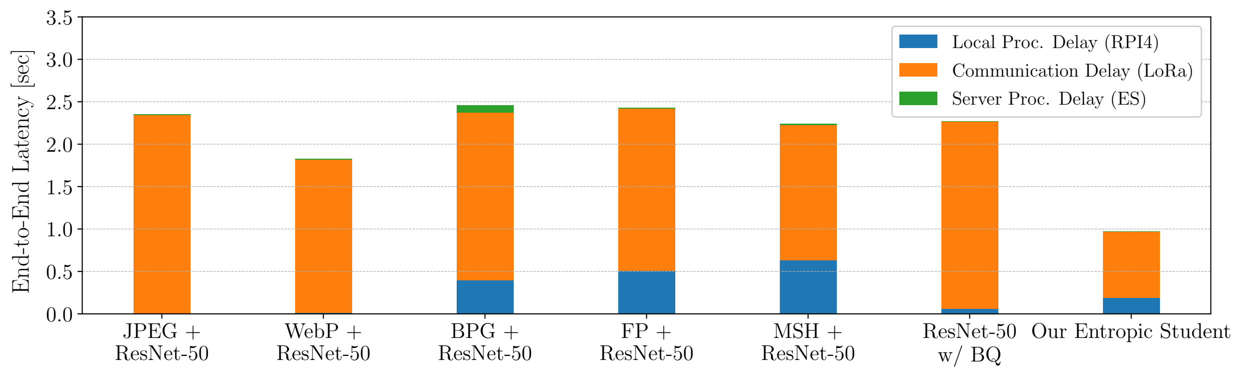

Figure S2: Component-wise delays to complete input-to-prediction pipeline, using RPI4 as mobile device.

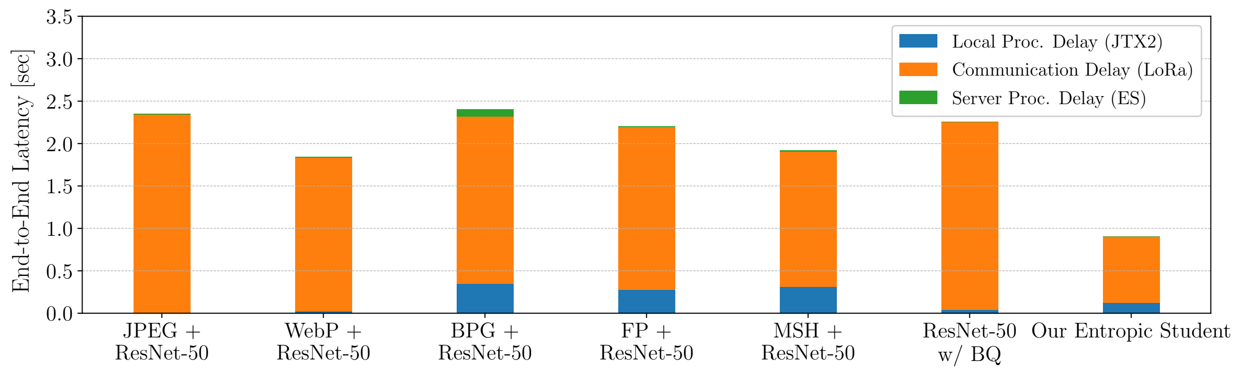

Figure S3: Component-wise delays to complete input-to-prediction pipeline, using JTX2 as mobile device.

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 *et al.* [26] evaluate the accuracy of their models on a 299×299 center crop. Since the pretrained ResNet-50 expects the crop size of 224×224,[4] we use the crop size for all the considered classifiers to highlight the effectiveness of our approach.

$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{cls}}(\hat{\mathbf{y}}, \mathbf{y})}_{\text{distortion}} - \beta \underbrace{\log p_{\phi}(f_{\theta}(\mathbf{x}) + \epsilon)}_{\text{rate}}, \quad \epsilon \sim \text{Unif}(-\tfrac{1}{2}, \tfrac{1}{2}) \tag{S7}$$

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 *et al.* train their models for 300k steps with batch size of 256 for 1.28M training samples, which is equivalent to 60 epochs ($= \frac{300k \times 256}{1.28M}$).

[4]`https://pytorch.org/vision/stable/models.html#classification`

[5]`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

[1] Eirikur Agustsson and Radu Timofte. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–135, 2017.

[2] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. Density Modeling of Images using a Generalized Normalization Transformation. In *International Conference on Learning Representations*, 2016.

[3] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end Optimized Image Compression. *International Conference on Learning Representations*, 2017.

[4] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[5] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. CompressAI: a PyTorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. `https://github.com/InterDigitalInc/CompressAI`.

[6] Fabrice Bellard. BPG Image format. `https://bellard.org/bpg/` [Accessed on August 6, 2021].

[7] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[8] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *2019 IEEE/ACM Int. Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[9] Google. Compression Techniques — WebP — Google Developers. `https://developers.google.com/speed/webp/docs/compression` [Accessed on August 6, 2021].

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

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Third International Conference on Learning Representations*, 2015.

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

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

[17] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*, 2017.

[18] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In *Proc. of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26, 2019.

[19] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems. *IEEE Access*, 8:212177–212193, 2020.

[20] Yoshitomo Matsubara and Marco Levorato. Neural Compression and Filtering for Edge-assisted Real-time Object Detection in Challenged Networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2272–2279, 2021.

[21] David Minnen, Johannes Ballé, and George D Toderici. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In *Advances in Neural Information Processing Systems*, pages 10771–10780, 2018.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[24] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2016.

[25] Jiawei Shao and Jun Zhang. BottleNet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2020.

[26] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Johannes Ballé, Abhinav Shrivastava, and George Toderici. End-to-end Learning of Compressible Features. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3349–3353. IEEE, 2020.

[27] George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and Challenge on Learned Image Compression (CLIC 2020), 2020.