Domain Adaptation for Learned Image Compression with Supervised Adapters

Domain Adaptation for Learned Image Compression with Supervised Adapters

Alberto Presta*, Gabriele Spadaro*, Enzo Tartaglione†,
Attilio Fiandrotti*† and Marco Grangetto*

*University of Turin
Computer science department
Turin, Italy
{name.surname}@unito.it

†LTCI, Télécom Paris,
Institut Polytechnique de Paris
Paris, France
{name.surname}@telecom-paris.fr

Abstract

In Learned Image Compression (LIC), a model is trained at encoding and decoding images sampled from a source domain, often outperforming traditional codecs on natural images; yet its performance may be far from optimal on images sampled from different domains. In this work, we tackle the problem of adapting a pre-trained model to multiple target domains by plugging into the decoder an adapter module for each of them, including the source one. Each adapter improves the decoder performance on a specific domain, without the model forgetting about the images seen at training time. A *gate network* computes the weights to optimally blend the contributions from the adapters when the bitstream is decoded. We experimentally validate our method over two state-of-the-art pre-trained models, observing improved rate-distortion efficiency on the target domains without penalties on the source domain. Furthermore, the gate's ability to find similarities with the learned target domains enables better encoding efficiency also for images outside them.

1 Introduction

Learned image compression (LIC) has gained considerable interest as it may achieve equal or even better compression efficiency than standardized codecs [1]. In this context, a parametric autoencoder is trained to project an image into a latent representation that is quantized and entropy-coded as a compressed bitstream. On the decoder side, the latent representation is decoded and projected back to the pixel domain, recovering the encoded image. The models suffer however a limited ability to adapt to images from a domain different than that of the training samples; standard video codecs include ad-hoc coding tools to deal with such contents [2], as a testament to their relevance. A straightforward approach to domain adaptation consists of fine-tuning the model over samples from the target domain. However, that comes at the risk of *catastrophic forgetting*, i.e. a performance loss on the source domain. In [3] image-specific adapter modules are plugged into the encoder, however, this method requires training the adapters at encoding time and delivering their parameters to the decoder for each single image, jeopardizing its practical deployability.
In this work, we propose a novel method for LIC domain adaptation based on blending the contributions of domain-specific adapters plugged at the decoder side. Instead of training a separate adapter for each image [3], an adapter module is plugged into

the decoder for both each target domain(s) and the source one. The adapters' contributions are blended according to the weights computed by a *gate* network that infers the domain of the image to encode. The adapters and the gate network are plugged into a pre-trained model that needs neither retraining nor refinement. The adapters and the gate are jointly trained and each adapter specializes in a specific domain, whether it is one of the target domains or the source one.
Our approach improves rate-distortion performance on the target domains, avoiding catastrophic forgetting on the source one. Furthermore, the adapters enhance image reconstruction even in classes that were never seen at training time. Besides being effective, adapters do not modify parameters related to the original pre-trained model; this ensures that even if these components were not available during decoding, the image would be reconstructed without compromising the performance of the original pre-trained model.

2 Background and related works

This section provides some background on LIC and reviews the relevant literature regarding domain adaptation in such a context.

2.1 Learned image compression

In LIC, a convolutional parametric autoencoder is trained end-to-end at compressing an image [1]. In a nutshell, the encoder receives an image $\mathbf{x}$ as input and projects it to a latent representation $\mathbf{y}$ that is quantized into $\hat{\mathbf{y}}$ and then entropy coded, yielding a compressed representation in the form of a bitstream. At the receiver side, a decoder projects this representation back to the original dimension, recovering an approximation of the original image $\hat{\mathbf{x}}$. The entire model is trained by optimizing a Rate-Distorsion loss function $\lambda D + R$, where D is the reconstruction error, R is the rate term, and λ is a hyperparameter that regulates the trade-off between the two. In seminal works like [4], the latent representation is modeled as a fully factorized distribution extracted either with an auxiliary neural network or in an analytical way [5]. [6] improved this approach by adding a *scale hyperprior* that aims to find spatial correlation within an image, using Gaussian priors. In [7], a local context model based on mask convolution is used to enhance entropy estimation, while [8] channel-wise contexts are exploited to decrease computing time. In the same wave, works like [9, 10] make use of local and window-based attention, while [11, 12] exploited a mixture of CNN and transformer. In general, the last models outperform conventional codecs on natural images; however, when transitioning to other specific domains, the emerging expected performance gap with traditional codecs narrows.

2.2 Task-domain adaptation for learned image compression

To the best of our knowledge, task-domain adaptation for LIC has not been exhaustively studied. In [13], the parameters related to the generalized divisive normalization layer [14] and the entropy model are fine-tuned, with the addition of a small number

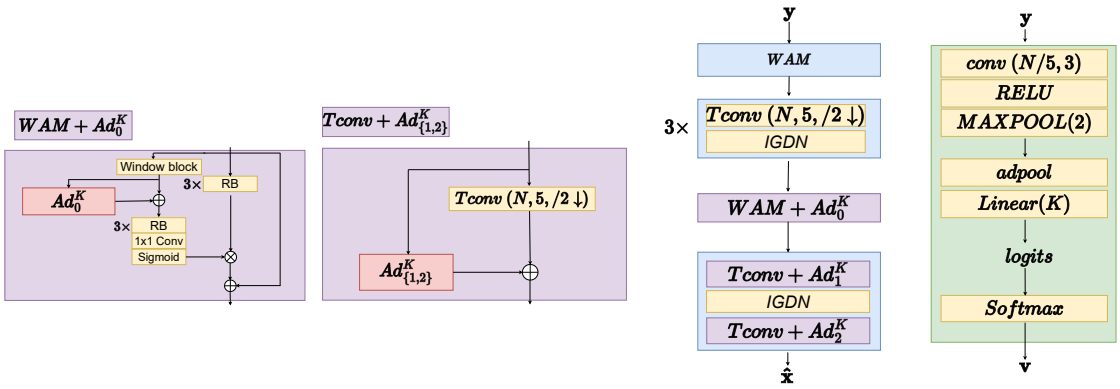
of custom channel-specific parameters; despite this strategy, the authors were unable to eradicate the problem of forgetting. Another possible approach to perform efficient domain adaptation is through the adapters [15], consisting of small modules added in the pre-trained model. In that sense, we identify [3] as the closest work to our method: Tsubota *et al.* exploit tiny adapters introduced on the second attention module in the decoder to adapt the model to a single image. Specifically, for each image, they optimize the latent representation rate-distortion-wise, and then they tune the adapter parameters minimizing a loss function that combines distortion and parameter rate. Despite its effectiveness, this work has some drawbacks. First, the parameters of the adapters are part of the bitstream (since they are specific for each image), which causes transmission overhead. Second, Tsubota *et al.* do not perform actual domain adaptation, but rather single-image content adaptation. Lastly, the encoding phase aligns with the optimization of the latent representation and adapter parameters, making the latter quite computationally expensive in terms of time and resources. On the other hand, we specialize adapters for multiple domains, either target or source, and we blend them to improve the final reconstruction, eliminating thus the need to encode adapter weights and simplifying the encoding phase, which remains unchanged compared to the model without adapters.

3 Proposed method and architecture

In this section, we describe our method and architecture for domain adaptation exemplifying the state-of-the-art Zou *et al.* [10] model. Yet, our method is architecture-agnostic as we experimentally show later on over the Cheng *et al.* [9] model. In a nutshell, we plug $K+1$ residual adapters into the decoder model pre-trained on a given source domain, i.e. K adapters are meant for novel target domains and one is for the legacy source domain. The adapter on the source domain could be skipped but it turns out to be useful for fine-tuning. At decoding, we exploited such adapters along with a gate network φ that yields a probability distribution over the $K+1$ domains $\mathbf{v} \in [0,1]^{K+1}$, that is used to blend adapters' outcomes.
The rest of this section is structured as follows. In Sec. 3.1 and Sec. 3.2 we first discuss the structure of the adapters and the gate, respectively. Next, in Sec. 3.3, we present the policy used to blend adapters' outputs. Finally, in Sec. 3.4, we describe the complete workflow of the resulting architecture along with the associated cost function and training process.

3.1 Domain adaptation at the decoder

In our architecture, one adapter $\mathbf{Ad}_{:}^{k}$ is plugged into the decoder for each of the $K+1$ domains of interest, i.e. K target domains plus the source one; Each adapter is composed by three modules $\mathbf{Ad}_{:}^{k} = \{Ad_0^k, Ad_1^k, Ad_2^k\}$, each one including one convolutional layer with 3×3 kernels. As shown in Fig. 1(a), the module Ad_0^k is plugged into the second Window Attention Module (WAM) block of the decoder, as in [3], to enhance the attention towards the more detailed parts of the image. Namely, the convolutional layer receives in input the output of the WAM block, preserving the



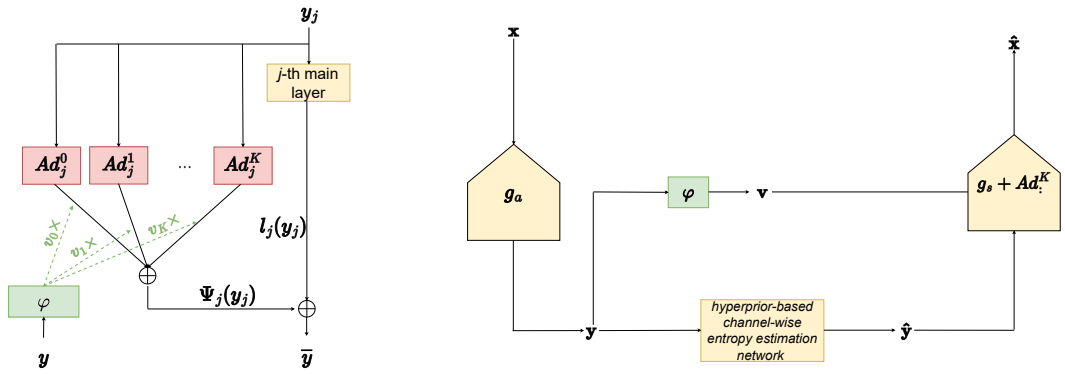
(a) *WAM adapters* (b) *Tconv. adapters* (c) *Decoder* (d) *Gate network* φ

Figure 1: (a) Architecture of the WAM adapter Ad_0^k. (b) Architecture of the deconvolutional adapters $Ad_{\{1,2\}}^k$. (c) Overall decoder. (d) Gate network φ.

number of channels and spatial dimensions. The remaining Ad_1^k and Ad_2^k modules are plugged at the output of the last two transpose-convolutional layers respectively, as in Fig. 1(b). For the sake of similarity, these two modules have the same structure as the respective layers they are plugged into, consisting of a deconvolutional layer that doubles the spatial dimensionality of the input feature maps. With respect to comparable literature where adapter modules are plugged into the encoder, our choice of plugging adapters into the decoder bears some advantages. First, the bitstream produced by the pre-trained reference encoder can be reused as it is. Second, training the adapters at the decoder avoids the cost of backpropagating the gradients to the encoder. The overall decoder is shown in Fig. 1(c).

3.2 Gate Network

The Gate network φ takes as input the latent representation $\mathbf{y}$ and outputs a domain probability distribution. The gate architecture is shown in Fig. 1(d) and includes one convolutional layer, a ReLU activation, a MaxPooling layer for complexity control, and an adaptive pooling layer that allows handling inputs of arbitrary size. A linear layer then yields $K+1$ classification logits that are normalized into the probability distribution $\mathbf{v} = (v_0, .., v_K)$ by a Softmax layer and delivered to the decoder. Such distribution amounts to $K+1$ real-valued coefficients that are compressed and embedded in the bitstream beside the compressed latent representation $\hat{\mathbf{y}}$. While the compression of such coefficients is out of the scope of this work, their rate can be safely assumed negligible to the compressed latent representation. For example, for a sample image from the classic Kodak dataset, the weight to encode $\mathbf{v}$ is on the order of 10^{-4} compared to the compressed latent representation. The gate could be in principle plugged into the decoder, sampling the compressed latent representation $\hat{\mathbf{y}}$. However, in this work we refrain our experiments from sampling the uncompressed latent representation $\mathbf{y}$, leaving the effect of quantizing the latent representation for



(a) *blending policy.* (b) *Overall LIC model architecture with φ (green box).*

Figure 2: (a) Adapters blending policy based on weighted sum. Index 0 refers to the source domain adapter. (b) Blueprint of a LIC model with the gate.

our future endeavors. Furthermore, using $\mathbf{y}$ as the input instead of the original image allows us to leverage the latent representation already extracted by the encoder, thus simplifying the gate's task.

3.3 Adapters blending policy

The outputs of the adapters are blended at the decoder according to the distribution coefficients $\mathbf{v}$ embedded in the bitstream. Namely, the probability distribution $\mathbf{v} = (v_0, .., v_K)$ among domains of interest, i.e. target and source, is used to compute a weighted average between the adapters outputs. Let $\mathbf{y}_j$ be the input of the j-th layer of the decoder with the adapter ($j = \{0, 1, 2\}$), adapters-enhanced output $\overline{\mathbf{y}}$ is calculated as

$$\overline{\mathbf{y}} = \Psi_j(y_j) + l_j(y_j), \tag{1}$$

where l_j represents the j-th layer of the decoder, and the addition operator is element-wise sum. Then we have

$$\Psi_j(y_j) = \sum_{k=0}^{K} v_k \cdot Ad_j^k(y_j), \tag{2}$$

where Ad_j^k is the j-th level adapter of the k-th domain. The blending policy just described is shown in Fig. 2(a), where we refer to the pre-trained layer network without adapter, as the main layer.

3.4 Training the adapters and the gate

This section first describes the workflow of our method and then how both adapters and the gate are trained given a model pre-trained on the source domain, e.g. natural images. The training set X_t is composed of T images uniformly distributed among the K target domains with the corresponding domain labels d_t, with $t = 1, ..., T$. As

shown in Fig. 2(b), an image $\mathbf{x}_t$ is passed to the encoder g_a producing the latent representation $\mathbf{y}$, which is first fed as input to the *gate network* φ that yields a probability distribution over the $K+1$ domains $\mathbf{v} \in [0,1]^{K+1}$. The probability vector $\mathbf{v}$ is then sent to the decoder beside the quantized latent representation $\hat{\mathbf{y}}$, which is encoded through the channel-wise entropy estimation module [8], and then it is used to weight the output of each adapter. Once decoded back, $\hat{\mathbf{y}}$ is passed to our adapter-based decoder $g_s + \mathbf{Ad}_{\cdot}^{K}$, returning the reconstructed image $\hat{\mathbf{x}}_\mathbf{t}$.
Once initialized from a Gaussian distribution, both the adapters $\mathbf{Ad}_{\cdot}^{k}$ and the gate φ are jointly trained for all domains of interest at once, freezing the other parameters of the model. The cost function minimized at training time is the sum between the reconstruction error D and a *domain mismatch term*, that should force φ to learn different domains from $\mathbf{y}$. By casting this as a classification problem, the objective is to minimize the cross-entropy (CE) between the distribution $\mathbf{v}$ derived from φ and the true labels d_t. Summing up, the minimized cost function is

$$\mathcal{L} = \gamma \cdot \mathrm{MSE}(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \mathrm{CE}(d_t, \mathbf{v}) \tag{3}$$

where γ is a hyperparameter that regulates the trade-off between these two terms, and MSE is the *mean squared error* minimizing the distortion. Notice that the second term in (3) does not represents the rate of $\hat{\mathbf{y}}$, typically minimized in the traditional RD loss function. Since we do not refine the encoder, the compressed latent representation $\hat{\mathbf{y}}$ remains untouched as well as its rate, allowing us to drop the rate component R from the minimized cost function.

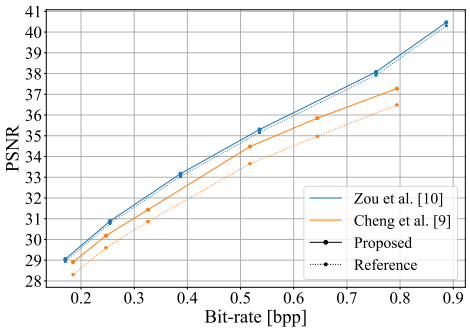
4 Experiments and Results

This section presents the results obtained with our method over the Zou *et al.* [10] and Cheng *et al.* [9] models; concerning the latter, we added the deconvolutional adapters in the last three layers of the decoder, used in the same way as for Zou *et al.* [10]. We experiment with adapting the pretrained models over two different target domains gauging the performance on the target as well as the source domain. Our implementation[1] leverages on the CompressAI library [16].

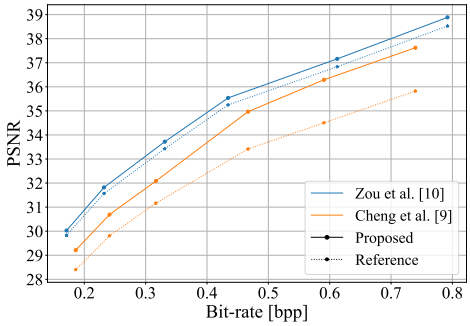
4.1 Experimental setup

We considered a total of $K=2$ target domains, namely the *Sketch* and *Comic* which are added to the source domain used for pretraining the model, e.g. natural images. Towards training the gate and the adapters, we had to combine different sources due to the lack of high-resolution datasets for domain adaptation in LIC. For the natural domain, we used the OpenImages dataset [17]; for the sketch domain, we used ImageNet-Sketch [18]; for the comic domain, we used BAM [19]. Namely, we randomly sampled 4k images from each dataset, resulting in a total of $T=12$k training images. During training, we froze all the parameters but those related to the adapters

[1] our code is available online at `https://github.com/EIDOSLAB/LIC-Domain-Adaptation-with-supervised-Adapters`.



(a) *Sketch*

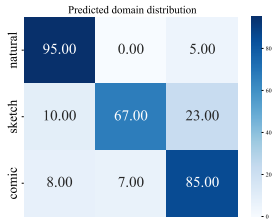


(b) *Comic*

Figure 3: Rate-PSNR plots for our Adapter-based method vs Reference for Zou *et al.* (blue) and Cheng *et al.* (orange), considering Sketch (a) and Comic (b).

Dataset	***Domain***	***Zou* et al.**		***Cheng* et al.**	
		BD-Rate	BD-PSNR	BD-Rate	BD-PSNR
Kodak	*Natural*	0.0012	~ 0	-1.46	0.06
clic	*Natural*	0.038	~ 0	-1.75	0.08
Sketch	*Sketch*	-2.45	0.1718	-11.55	0.69
Comic	*Comic*	-4.93	0.28	-19.15	1.25

(a) BD-Rate/PSNR for different domains.



(b) Avg. probabilities predicted by φ.

Figure 4: (a) BD-Rate/PSNR for different domains, considering both Zou *et al.* and Cheng *et al.* as anchor, with our models as reference. (b) Avg. probabilities $\mathbf{v}$ predicted by the Gate φ, starting from Zou *et al.* for the second-best quality model.

and the gate, i.e. the pre-trained encoder and decoder are not refined. We trained gate and adapters for 400 epochs using Adam with an initial learning rate of 10^{-4}, halving it when reaching a plateau with the patience of 15 epochs and batch size of 16 and with $\gamma = 0.5$ in (3). At inference time, we assess the performance of our method over four distinct test datasets. The target sketch and comic domains are represented by 100 random images from ImageNet-Sketch and BAM that are not present in the training set. The source domain is represented by the Kodak [20] and CLIC [21].The performance of our method is reported in terms of RD curves, BD-Rate, and BD-PSNR. Concerning complexity, the adapter modules include about 3.8M parameters, i.e. $\sim 5.5\%$ of Zou *et al.* decoder complexity. Similarly, the gate is composed of 184k parameters, i.e. $\sim 0.28\%$ of Zou *et al.* encoder complexity.

4.2 Rate-Distortion Performance

Fig. 3 shows the RD performance of our method for the Zou *et al.* and Cheng *et al.* models over the target sketch and comic domains. Our method visibly improves the performance for both the target domains to the pre-trained model. Table 4a summarizes the two models' performance in terms of BD-Rate and BD-PSNR gains over the four test datasets. Our method strikes rate reductions of 5% and 10% over the comic and sketch domains. Yet, we achieve some gain also over the source

domain(Kodak and Clic datasets) for Cheng *et al.*, proving our domain adaptation method does not incur in any *catastrophic forgetting*. The BD-Rate reduction is remarkable Especially for Cheng *et al.*: we hypothesize this model is less able to generalize outside the source domain than Zou *et al.*, hence our measured gains. Fig. 4b shows how the gate network blends the outputs of the adapters across different domains for Zou *et al.*. For all domains the adopted training policy induces the exploitation of all the available adapters to increase the performance since; even though the gate is capable of identifying the predominant domain for an image, it still utilizes the others to a lesser extent. Fig. 5 shows samples of decoded images: our method preserves best the fine-grained details over the target domains.

(a) Original Image (b) Zou *et al.* PSNR: 25.62 (c) Prop. PSNR: 25.90

Figure 5: Reconstruction of a sample taken from BAM [19] (from Comic domain). (a) Original, (b) Zou *et al.*, (c) Proposed. Both (c) and (d) have bpp equal to 0.31.

Dataset	***Reference***	***Blending method***		
		Proposed	Top1	Oracle
Kodak (Natural)	35.947	35.945	35.948	**35.950**
Clic (Natural)	36.618	36.613	36.614	**36.618**
Sketch	37.925	**38.112**	38.005	38.109
Comic	36.833	**37.161**	37.138	37.141

Table 1: PSNR on different test sets considering different blending policies.

4.3 Comparison with other blending policies

Next, we assess the impact of the policy used to blend the adapters modules outputs. We recall that with our *proposed* method, the outputs of the adapters are weighted according to the probability distribution predicted by the gate network φ. We now consider two other blending approaches, namely *top1* and *oracle*. The former considers only the output of the adapter corresponding to the top-scoring class predicted by the gate, dropping the contributions from other adapters. The latter is a hypothetical scheme that relies on the assumption that the image class label is known at testing time, discarding any input from the gate. We highlight that these policies are used not only at inference but also during the training phase. Tab. 1 shows the results in terms of PSNR considering the second highest quality model in Fig. 3. The *proposed* policy outperforms the other two for both target domains, meaning that exploiting all the adapters with the right blending maximizes the image quality. On the other hand, the oracle policy yields the best image quality over natural images, i.e. the

source domain. However, the oracle policy relies on the hypothesis that the image domain is known for each image to encode, which is unrealistic in most cases.

4.4 Unseen domains

We evaluate our methods for domains unseen both when pretraining the Zou *et al.* model and at the time of training the adapters and the gate over the two target domains. We considered different types of images from different datasets like BAM [19], DomainNet [22], and IIIT-AR-13K [23]. Tab. 2 shows that for every domain our method yields better encoding efficiency, even if sometimes minimally. However, for some domains the gains are remarkable; for example, the 20% gain for *Quickdraw* is likely to be attributed to its semantic similarity to the *Sketch* domain. This hypothesis is supported by the probability distribution generated by the gate network φ, where approximately 90% of the image likelihood is assigned to the adapter associated with the Sketch domain. For its possible practical application, an important result is the BD-Rate equal to -2.45 on [23], representing graphical and documents images: in this case, we see how φ associated this domain more with the Sketch class, which is intuitive concerning the contents of this dataset. In general, The addition of domain-specific adapters also brings further performance improvement, and therefore knowledge, in other typologies of images, thanks to increased generalization capacity and the capacity of the gate to spot correlations with known domains.

Dataset	***Bjontegaard Metrics***		***Predicted domain distribution***		
	BD-Rate	BD-PSNR	Natural	sketch	comic
Infograph [22]	-0.44	0.007	48	3	49
Bam-Drawing [19]	**-2.789**	0.216	12	42	46
Quickdraw [22]	**-21.574**	1.95	0	93	7
watercolor [19]	-0.61	0.02	49	4	17
clipart [22]	**-1.32**	0.065	15	0.07	78
IIIT-AR-13K [23]	**-2.45**	0.20	15	50	35

Table 2: *First two columns*: Bjontegaard Metrics for unseen domains, considering [10] as reference. *Second three columns*: average **v** considering highest quality model.

5 Conclusions and future works

This study addressed the problem of domain adaptation in learned image compression. Our method outperforms reference pre-trained state-of-the-art models on the target domains without forgetting the source one (Sec. 4.2), showing also (Sec. 4.3) that aggregating the adapters brings a slight performance improvement. In Sec. 4.4 we showed that our method offers advantages also over unseen domains.This method could benefit from improvements like unsupervised gate training to remove the need for predefined adapter classes. Additionally, domain adaptation at the encoding stage could enhance performance by modifying the entropy estimation.

Acknowledgements

This work was partially funded by the Hi!PARIS Center on Data Analytics and Artificial Intelligence.

References

[1] Siwei Ma et al., "Image and video compression with neural networks: A review," *Transactions on Circuits and Systems*, 2019.

[2] Jizheng Xu et al., "Overview of the emerging hevc screen content coding extension," *IEEE Transactions on Circuits and Systems for Video Technology*, 2015.

[3] Koki Tsubota et al., "Universal deep image compression via content-adaptive optimization with adapters," in *IEEE W. C. on Applications of Computer Vision*, 2023.

[4] Johannes Ballé et al., "End-to-end optimized image compression," in *5th International Conference on Learning Representations*, 2017.

[5] Alberto Presta et al., "A differentiable entropy model for learned image compression," in *International Conference on Image Analysis and Processing*, 2023.

[6] Johannes Ballé et al., "Variational image compression with a scale hyperprior," in *International Conference on Learning Representations (ICLR)*, 2018.

[7] David Minnen et al., "Joint autoregressive and hierarchical priors for learned image compression," *Advances in neural information processing systems*, 2018.

[8] David Minnen and Saurabh Singh, "Channel-wise autoregressive entropy models for learned image compression," in *International Conference on Image Processing)*, 2020.

[9] Zhengxue Cheng et al., "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020.

[10] Renjie Zou et al., "The devil is in the details: Window-based attention for image compression," in *IEEE conference on computer vision and pattern recognition*, 2022.

[11] Ming Lu et al., "Transformer-based image compression," in *2022 Data Compression Conference (DCC)*. IEEE, 2022.

[12] Jinming Liu et al., "Learned image compression with mixed transformer-cnn architectures," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2023.

[13] Sudeep Katakol et al., "Danice: Domain adaptation without forgetting in neural image compression," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2021.

[14] Johannes Ballé et al., "Density modeling of images using a generalized normalization transformation," in *4th International Conference on Learning Representations*, 2016.

[15] Yi-Lin Sung et al., "Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2022.

[16] Jean Bégaint et al., "Compressai: a pytorch library and evaluation platform for end-to-end compression research," *arXiv preprint arXiv:2011.03029*, 2020.

[17] Alina Kuznetsova et al., "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[18] Haohan Wang et al., "Learning robust global representations by penalizing local predictive power," in *Advances in Neural Information Processing Systems*, 2019.

[19] Michael Wilber et al., "Bam! the behance artistic media dataset for recognition beyond photography," in *IEEE international conference on computer vision*, 2017.

[20] Rich Franzen, "Kodak lossless true color image suite," *source: http://r0k.us/graphics/kodak*, vol. 4, no. 2, pp. 9, 1999.

[21] George Toderici et al., "Workshop and challenge on learned image compression (clic2020)," in *CVPR*, 2020.

[22] Xingchao Peng et al., "Moment matching for multi-source domain adaptation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[23] Ajoy Mondal et al., "Iiit-ar-13k: new dataset for graphical object detection in documents," in *14th IAPR International Workshop*, 2020.