Slimmable Compressive Autoencoders for Practical Neural Image Compression

Fei Yang[1,2], Luis Herranz[2], Yongmei Cheng[1], Mikhail G. Mozerov[2]
[1] School of Automation, Northwestern Polytechnical University, Xi'an, China
[2] Computer Vision Center, Universitat Autonoma de Barcelona, Barcelona, Spain
{fyang,lherranz,mozerov}@cvc.uab.es, chengym@nwpu.edu.cn

Abstract

Neural image compression leverages deep neural networks to outperform traditional image codecs in rate-distortion performance. However, the resulting models are also heavy, computationally demanding and generally optimized for a single rate, limiting their practical use. Focusing on practical image compression, we propose slimmable compressive autoencoders (SlimCAEs), where rate (R) and distortion (D) are jointly optimized for different capacities. Once trained, encoders and decoders can be executed at different capacities, leading to different rates and complexities. We show that a successful implementation of SlimCAEs requires suitable capacity-specific RD tradeoffs. Our experiments show that SlimCAEs are highly flexible models that provide excellent rate-distortion performance, variable rate, and dynamic adjustment of memory, computational cost and latency, thus addressing the main requirements of practical image compression.

Method	Rate-dist. perform.	Total memory	Variable			Training time
			Rate	Memory	FLOPs	
JPEG,JP2K	Low	Very low	Yes	-	-	-
BPG	High	Low	Yes	-	-	-
Single CAE	Optimal	Medium	No	No	No	Low
Multiple CAEs	Optimal	High	Yes	Yes	Yes	High
BScale[36]	Medium	Medium	Yes	No	No	Low
MAE[42],cAE[10]	High	Medium	Yes	No	No	Low
SlimCAE	Optimal	Medium	Yes	Yes	Yes	Low

Table 1: Comparison of compression methods.

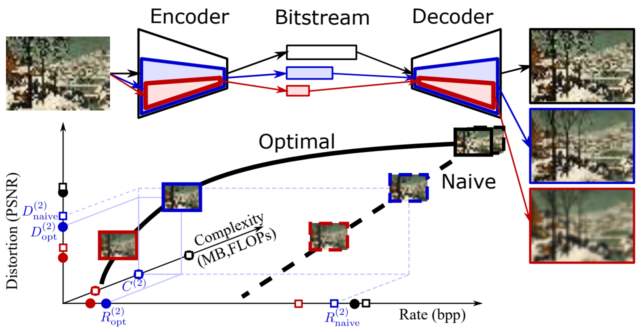


Figure 1: Variable rate and complexity adaptive image compression with a slimmable compressive autoencoder.

1. Introduction

Visual information (e.g. images, videos) plays a central role in human content creation, communication and interaction. Efficient storage and transmission through constrained channels requires compression. We can thus consider the *basic (lossy) image compression* problem, where the goal is to obtain the shortest binary representation (i.e. lowest *rate* bitstream) that can represent the input image at a certain level of fidelity (i.e. minimum *distortion*). Thus, low rate and low distortion are fundamentally opposing objectives, in practice involving a *rate-distortion (RD) tradeoff*. The more challenging problem of *practical image compression* further includes real-world constraints such as memory, computation and latency, related with their implementation in resource-constrained devices (e.g. mobile phones) and networks. Similarly, video compression addresses the same problem for sequences of images, where low complexity and latency become even more critical [41]. Many applications also require dynamic control of the RD tradeoff to adapt to specific rate requirements (i.e. *variable rate*).

Traditional methods (e.g. JPEG [40], JPEG2000 [25, 31], BPG [32]) follow the transform coding paradigm with carefully designed linear transforms and coding tools, to effectively address practical image compression. Encoding is block-based and iterative. RD is optimized during encoding by exhaustively searching optimal block partitions and coding and prediction modes. Complexity can be controlled by limiting the range of coding and prediction modes, or using heuristics. The rate can be estimated from previous blocks, and controlled by adjusting quantization parameters.

A more recent paradigm is *neural image compression* (NIC) [37, 6, 11, 38, 36, 15, 20, 21, 22, 19] (or learned image compression), which exploits flexible nonlinear transforms and entropy models parametrized as deep neural networks. The framework typically consists of an autoencoder followed by quantization and entropy coding (henceforth a *compressive autoencoder* -CAE- [36]). While they can outperform traditional image codecs in RD, they are typically only optimal for a single target rate, and at the cost of much heavier and computationally expensive models that require specialized hardware (e.g. GPUs), making them unattractive in practical resource-limited scenarios. In contrast to block-based codecs, NIC approaches are typically image-based and feed-forward, resulting in a constant processing cost. While some approaches address variable rate [36, 8, 42, 10], practical concerns related with efficiency still remain largely unaddressed (see Table 1).

The challenge of deploying deep neural networks in resource-limited devices (e.g. smartphones, tablets) has motivated research on lightweight architectures [12, 29], integer and binary networks [16, 27, 13] and automatic architecture search [34]. However, only few works have addressed efficiency in NIC [28, 14]. We borrow the idea of slimmable neural networks [44], where the width of the layers (i.e. number of channels) of a classifier can be adjusted to trade off accuracy for computational efficiency.

In this paper, we propose the slimmable compressive autoencoder (SlimCAE) framework, where we show that the slimming mechanism can enable both variable rate and adaptive complexity (see Fig. 1). We propose and study different variants of slimmable generalized divisive normalization [5] (GDN) layers, and slimmable probability models. Naive training of SlimCAEs, with the different subnetworks (i.e. *subCAEs*) optimizing the same loss on all widths, results in suboptimal performance. We crucially observe that each RD tradeoff has an corresponding minimum capacity. This suggest that, in contrast to other slimmable networks, each width should have different objectives, i.e. different $D + \lambda R$, determined by the corresponding tradeoff λ. This characteristic makes SlimCAEs more difficult to train, and unlikely to benefit from implicit or explicit distillation [44]. Addressing this problem, we propose λ-scheduling an algorithm that alternates between training the model and adjusting the different λs. Via slimming, SlimCAEs can address the main requirements of *practical neural image compression (PNIC)* in a simple and integrated way.

Our main contributions are: (1) a novel rate and complexity control mechanism via layer widths, motivated by a key insight connecting optimal RD tradeoffs and capacity; (2) the SlimCAE framework, which enables control of computation, memory and rate, required for PNIC; (3) an efficient training algorithm for SlimCAEs; (4) novel slimmable modules (i.e. GDNs, entropy models). In addition, SlimCAEs can be easily adapted to obtain scalable bitstreams.

2. Related work

2.1. Neural image compression

The modern non-linear deep autoencoding framework with quantization and entropy coding (which here we refer to as compressive autoencoder [36]), trained by backpropagation to minimize a (fixed) combination of rate and distortion, is relatively recent [37, 36, 6]. Encoders and decoders often integrate multi-scale structures [8, 28, 24] and generalized divisive normalization (GDN) layers [5, 6]. End-to-end training requires replacing non-differentiable quantization by differentiable proxies such as additive noise [6], identity in the backward pass [36] and soft-to-hard vector quantization [1]. Entropy coding also benefits from learnable CNNs, via hyperpriors [7] and contextual models [21, 17, 18, 22, 23]. More recently, adversarial training has been used to target very low rates [28, 39, 2].

2.2. Variable rate image compression

Many practical applications require certain control of the target rate. Traditional image compression methods enable this functionality via quantization tables that scale DCT coefficients according to the target rate. Similar to traditional methods, Theis *et al.* [36] learns a set of rate-specific parameters to scale the bottleneck feature before quantization (we refer to this as *bottleneck scaling*, see Fig. 2a). Modulated autoencoders (MAEs) [42] and conditional autoencoders (cAEs) [10] show that modulating also intermediate features improves RD performance (see Fig. 2b). Recurrent neural networks can also realize variable rate coding [37], yet are demanding computationally. Cai *et al.* [8] proposed a multi-scale decomposition network, each scale targeting a different rate. None of these methods provides explicit control over complexity. Our approach (see Fig. 2c), in contrast, can jointly reduce significantly the memory and computational cost for low rates.

2.3. Efficiency

Lightweight architectures, such as GoogleNet [33] and MobileNet [12, 29], are designed for resource-limited devices by reducing the number of parameters and computation. At the cost of small drop in performance, integer or binary weights can further improve efficiency [16, 27, 13]. Network architecture search (NAS) [45, 3, 34] includes design hyperparameters (e.g. width, number of layers) in the optimization space. Slimmable neural networks [44, 43] enable models that can be run at different accuracy-efficiency tradeoffs. Regarding NIC, Johnston *et al.* [14] use NAS to achieve 2-3× coding speed-up. Cai *et al.* [9] use progressive coding to reduce initial latency, although memory and computational cost remain similar. While tackling run-time or latency, these methods still focus on a single RD tradeoff, not providing rate, memory nor computation control.

3. Slimmable compressive autoencoders

3.1. Slimmable autoencoders

The basic structure of an autoencoder (AE) is a learnable encoder $\mathbf{z} = f(\mathbf{x}; \theta)$ parametrized by θ that maps an

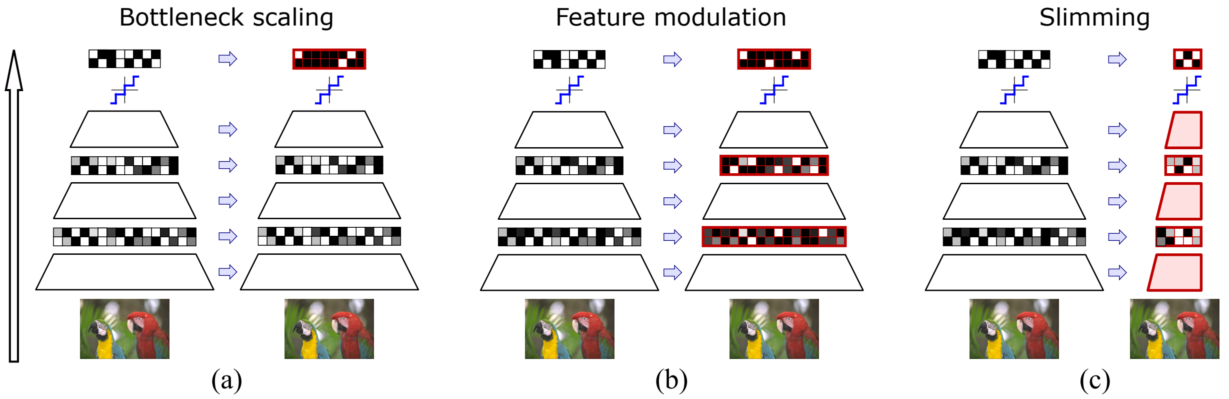


Figure 2: Mechanisms to achieve variable rate in compressive autoencoders: (a) bottleneck scaling [36], (b) feature modulation [42, 10], and (c) proposed method (SlimCAE). Adaptation from high rate (left) to low rate (right). Changes are highlighted in red. Only SlimCAE reduces memory and computation. GDN layers are not included for simplicity.

input image $\mathbf{x} \in \mathbb{R}^N$ to a transformed (latent) representation $\mathbf{z} \in \mathbb{R}^D$, followed by a learnable decoder $\hat{\mathbf{x}} = g(\mathbf{z}; \phi)$ parametrized by ϕ that maps the latent representation to $\hat{\mathbf{x}} \in \mathbb{R}^N$, with the objective of reconstructing the input image $\mathbf{x}$. Hence the combination of encoder and decoder is autoencoding $\mathbf{x}$, and the objective is to learn the parameters $\psi = (\theta, \phi)$ by minimizing a loss $\mathcal{L}(\theta, \phi; \mathcal{X})$ given a training dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{|\mathcal{X}|}$. The loss measures the reconstruction error, possibly combined with other objectives.

We are interested in AEs whose layers are slimmable [44], i.e. *slimmable autoencoders* (*SlimAEs*), thus enabling dynamic control over the memory and computation costs. An *slimmable layer* allows for discarding part of the layer parameters (in most cases is equivalent to setting them to zero) while still performing a valid operation, trading off expressiveness for lower memory and computational cost. We consider SlimAEs containing K *subautoencoders* (*subAEs*), each of them parametrized by a pair $\psi^{(k)} = \left(\theta^{(k)}, \phi^{(k)}\right) \in \Psi = \left\{\left(\theta^{(1)}, \phi^{(1)}\right), \ldots, \left(\theta^{(K)}, \phi^{(K)}\right)\right\}$, where we assume that $\theta^{(1)} \subset \cdots \subset \theta^{(K)} = \theta$ and $\phi^{(1)} \subset \cdots \subset \phi^{(K)} = \phi$ (we assume these conditions are met for every layer in the SlimAE). Similarly, we can define the loss for the subAE k as $\mathcal{L}^{(k)}\left(\theta^{(k)}, \phi^{(k)}; \mathcal{X}\right)$, and train the SlimAE with the joint loss or a weighted average $\mathcal{L}(\Psi; \mathcal{X}) = \sum_k \mathcal{L}^{(k)}\left(\theta^{(k)}, \phi^{(k)}; \mathcal{X}\right)$.

3.2. Compressive autoencoders

A compressive autoencoder (CAE) is an AE, where the output of the encoder is a binary stream (*bitstream*), typically stored or transmitted through a communications channel. The objective is to maximize the quality of the reconstructed image (i.e. minimize the *distortion*) while minimizing the number of bits transmitted (i.e. minimize the *rate*). CAEs are based on AEs, where the encoder is followed by a quantizer $\mathbf{q} = Q(\mathbf{z})$, where $\mathbf{q} \in \mathbb{Z}^D$ is a discrete-valued symbol vector. A lossless entropy encoder then binarizes

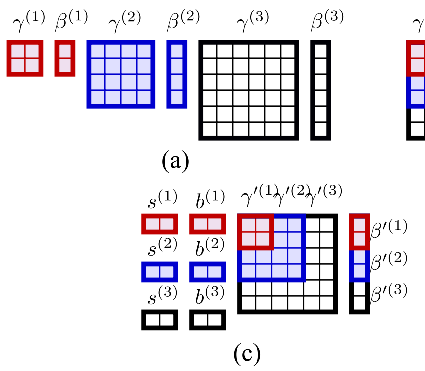


Figure 3: GDN variants: (a) SwitchGDN, (b) SlimGDN, (c) SlimGDN+ (SlimGDN with switch. param. modulation).

and serializes $\mathbf{q}$ into the bitstream $\mathbf{b}$, exploiting its statistical redundancy to achieve code lengths close to its entropy. These operations are reversed in the decoder.

CAEs are typically trained by solving a *rate-distortion optimization (RDO)* problem with loss

$$\mathcal{L}(\theta, \phi; \mathcal{X}, \lambda) = D(\theta, \phi; \mathcal{X}) + \lambda R(\theta; \mathcal{X}), \tag{1}$$

where $\mathcal{X}$ is the training dataset, λ is the (fixed) tradeoff between rate and distortion. To allow end-to-end optimization with backpropagation, during training non-differentiable operations, such as quantization and entropy coding, are replaced by differentiable proxies, such as additive noise and entropy estimation.

Without loss of generality, we focus on the CAE framework of Balle *et al.* [6], which combines convolutional layers, generalized divisive normalization (GDN) and inverse GDN (IGDN) layers, scalar quantization to the nearest neighbor (i.e. i.e. $\mathbf{q} = \lfloor \mathbf{z} \rceil$) and arithmetic coding. During training, quantization is replaced by additive uniform noise (i.e. $\tilde{\mathbf{z}} = \mathbf{z} + \Delta\mathbf{z}$, with $\Delta\mathbf{z} \sim \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$). Similarly, arithmetic coding is bypassed and rate is approximated by the entropy of the quantized symbol vector $R(\mathbf{b}) \approx H[P_{\mathbf{q}}] \approx H[p_{\tilde{\mathbf{z}}}(\tilde{\mathbf{z}}; \nu)]$, where ν are the parameters of the entropy model used in [6]. Distortion is measured as the reconstruction mean square error (MSE), i.e. $\|\mathbf{x}-\hat{\mathbf{x}}\|^2$. The CAE is thus parametrized by $\psi=(\theta,\phi,\nu)$.

3.3. Slimmable CAEs

In order to obtain a *slimmable compressive autoencoder* (*SlimCAE*), all operations in the CAE are required to be non-parametric, slimmable or efficiently switchable. Quantization is non-parametric in our case, and convolutional layers are implemented slimmable [44]. For GDN/IGDN layers, we propose and compare several variants (see next subsection) layers. Finally, we use switchable entropy models, i.e. each subCAE k has its own parameters $\nu^{(k)}$, which can be easily switched since the size is negligible compared to the other parameters $\theta^{(k)}$ or $\phi^{(k)}$.

Our approach can also be extended to more complex frameworks including hyperpriors [7] and autoregressive context models [22].

3.4. Switchable and slimmable GDN/IGDN layers

While GDN [4] was proposed to Gaussianize the local joint statistics of natural images, Balle *et al.* [6] proposed an approximate inverse operation (IGDN), and showed that GDN/IGDN layer pairs are highly beneficial in learned image compression, and since then have been adopted by many CAE frameworks. Both GDN and IGDN are parametrized by $\gamma\in\mathbb{R}^{w\times w}$ and $\beta\in\mathbb{R}^{w}$, where w is the number of input (and output) channels.

In the case of a SlimCAE with K subCAEs, the input to the GDN layer has the following possible channel dimensions $w^{(1)},\ldots,w^{(K)}$. We consider three possible variants:

- **Switchable GDNs**[1]. We use independent sets of parameters $\gamma^{(k)}\in\mathbb{R}^{w^{(k)}\times w^{(k)}}$ and $\beta^{(k)}\in\mathbb{R}^{w^{(k)}}$ for every subGDN k (see Fig. 3 a). The normalized representation for an input $\mathbf{y}^{(k)}\in\mathbb{R}^{w^{(k)}}$ is then

$$\tilde{y}_i^{(k)}=\frac{y_i^{(k)}}{\left(\beta_i^{(k)}+\sum_j\gamma_{ij}^{(k)}|y_j^{(k)}|^2\right)^{\frac{1}{2}}} \tag{2}$$

  While flexible, the total number of parameters is relatively high $\sum_{k=1}^{K}\left(w^{(k)}+1\right)w^{(k)}$, and switching may be not very efficient.

- **Slimmable GDN (SlimGDN)**. A more compact option is to reuse parameters from smaller subGDNs by imposing $\gamma^{(1)}\subset\cdots\subset\gamma^{(K)}$ and $\beta^{(1)}\subset\cdots\subset\beta^{(K)}$. Now the total number of parameters in a SlimGDN layer is $\left(M^{(K)}+1\right)w^{(K)}$ (see Fig. 3b).

- **SlimGDN with switchable parameter modulation**. SlimGDNs usually performs worse than switchable GDNs, since they are less flexible to adapt to the statistics of the different $\mathbf{y}^{(k)}$. We propose a variant using switchable parameter modulation, where a global scale and bias are learned separately for every subGDN (i.e. switchable), i.e. $\gamma_{ij}^{(k)}=s_\gamma^{(k)}\gamma'^{(k)}_{ij}+b_\gamma^{(k)}$ and $\beta_i^{(k)}=s_\beta^{(k)}\beta'^{(k)}_i+b_\beta^{(k)}$, where $\gamma'^{(k)}$ and $\beta'^{(k)}$ are shared and slimmable and $s_\gamma^{(k)}$, $b_\gamma^{(k)}$, $s_\beta^{(k)}$ and $b_\beta^{(k)}$ are switchable scalars specific for the subGDN k. This variant requires only 4 additional parameters per subGDN (see Fig. 3c), for a total number of parameters $\left(w^{(K)}+1\right)w^{(K)}+4K$.

[1]In the following, we omit IGDN for clarity (the same analysis applies).

3.5. (Naive) training of SlimCAEs

We can extend Eq. (1) and optimize the joint loss of all K subCAEs $\arg\min_\psi\sum_{\psi\in\Psi}\mathcal{L}(\psi;\mathcal{X},\lambda)$, with parameters $\Psi=\left\{\psi^{(1)},\ldots,\psi^{(K)}\right\}$. The problem can be solved using stochastic gradient descent (SGD) and backpropagation. We refer to this case as *naive SlimCAE*.

4. SlimCAEs with multiple rate-distortion tradeoffs

4.1. Rate-distortion and capacity

While a naive SlimCAE can already control the rate of the output bitstream and the complexity of the model, it is limited to a relatively narrow range of rates with suboptimal RD performance. This can be observed in Fig. 4, where we show the RD curves obtained with independent CAEs with different capacities controlled via the layer width w (i.e. number of filters per convolutional layer) compared with one SlimCAE with the same layer widths.

Fig. 4 shows that there is a limit in the minimum achievable distortion by a CAE (i.e. at high rates), which is in turn related to its capacity (the lower the capacity, the higher the minimum distortion). Additionally, the figure shows that, when the rate is low enough, additional capacity is unnecessary since the curves converge before that point, (i.e. an optimal capacity for every segment of the optimal RD curve).

Note also that the RD points of the SlimCAE are located over the RD curves of independent CAEs with the same capacity, suggesting that a slimmable version does not entail RD penalty. We aim at training the SlimCAE so it can achieve optimal RD performance at the different capacities. We define the *multi-RD optimization* (MRDO) loss as

$$\mathcal{L}(\Psi;\mathcal{X},\Lambda)=\sum_{k=1}^{K}D\left(\psi^{(k)};\mathcal{X}\right)+\lambda^{(k)}R\left(\psi^{(k)};\mathcal{X}\right), \tag{3}$$

where $\Lambda=\left\{\lambda^{(1)},\ldots,\lambda^{(K)}\right\}$ is a given set of RD tradeoffs for the different K subCAEs, and the corresponding MRDO problem is $\arg\min_\Psi\sum_{\psi^{(k)}\in\Psi}\mathcal{L}(\Psi;\mathcal{X},\Lambda)$.

An important aspect to note is that in this case each subCAE solves a different optimization problem determined by the specific tradeoff $\lambda^{(k)}$. This is an important difference with slimmable networks and SlimAEs in general (including naive SlimCAEs), where every subnetwork or subAE solves exactly the same problem, just with different capacity. This has implications, such as more difficulty to jointly solve all the subproblems and also makes implicit distillation [44] across subCAEs more unlikely. The problem now is to find appropriate values of Λ and train the SlimCAE.

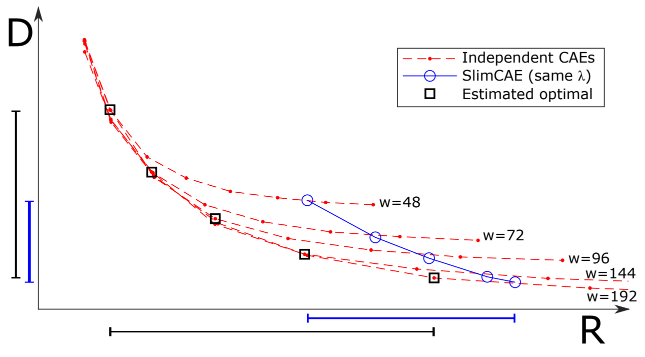


Figure 4: Comparison of RD curves of independent CAEs and SlimCAE with shared λ. Better choices in the RD curves are also highlighted.

4.2. Estimating optimal λs from RD curves

One possible way is to leverage the RD curves of independent CAEs and try to estimate the optimal points to switch to the next subCAE, which is where the curves start diverging because RD performance saturates for that capacity. We can then estimate $\lambda^{(k)}$ as the slope of the corresponding curve at that optimal point (see Fig. 4).

4.3. Automatic estimation via λ-scheduling

While knowing in advance the empirical RD curves leads to better RD performance and wider rate ranges, it has the important drawback of having very high computation cost, since we need to compute K RD curves, one for each target capacity, and every curve requires training a number of independent CAE exploring different λs.

MRDO training with λ scheduling In order to address the previous limitation, we propose an effective MRDO training algorithm to automatically estimate Λ without requiring independent CAEs curves (see Alg. 1).

The algorithm is based on progressively varying the values of every $\lambda^{(k)}$ following a predefined schedule. We alternate phases of updating Λ and updating the SlimCAE using SGD. The initial stage is a naive SlimCAE, where λ is set to target high rate and low distortion, which requires full use of the capacity. Once trained, the SlimCAE is already optimized for that full capacity. We fix $\lambda^{(K)}$ and update the remaining $k = 1, \ldots, K-1$ as $\lambda_t^{(k)} = \kappa \lambda_{t-1}^{(k)}$ with a factor $\kappa > 1$. Then we update the SlimCAE for another number

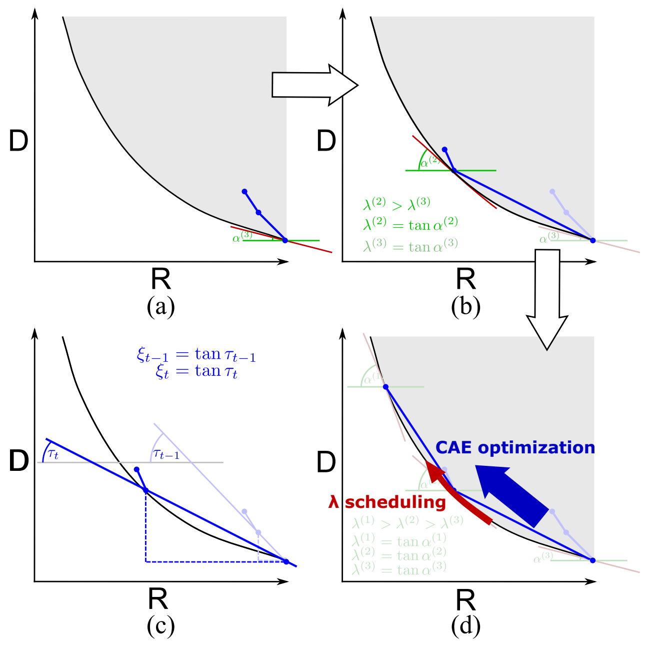

D
R
(a)
$\alpha^{(3)}$
D
R
(b)
$\alpha^{(2)}$
$\lambda^{(2)} > \lambda^{(3)}$
$\lambda^{(2)} = \tan\alpha^{(2)}$
$\lambda^{(3)} = \tan\alpha^{(3)}$
$\alpha^{(3)}$
D
R
(c)
$\xi_{t-1} = \tan\tau_{t-1}$
$\xi_t = \tan\tau_t$
τ_t
τ_{t-1}
D
R
(d)
CAE optimization
λ scheduling
$\lambda^{(1)} > \lambda^{(2)} > \lambda^{(3)}$
$\lambda^{(1)} = \tan\alpha^{(1)}$
$\lambda^{(2)} = \tan\alpha^{(2)}$
$\lambda^{(3)} = \tan\alpha^{(3)}$

Figure 5: Training SlimCAEs: (a) naive training with a single RD tradeoff λ leads to small ranges and suboptimal RD performance, (b) varying the λ progressively for smaller subCAEs changes the target RD point and stretches the RD range, (c) the slope ξ of RD segments is used to monitor convergence of the proposed training algorithm, and (d) illustration of how λ scheduling changes the RD target and SlimCAE optimization stretches the RD range.

of iterations, which tends to reduce the rate and moves the RD of subCAE $K-1$ closer to the optimal RD curve. Geometrically, this results in the slope of the segment between the RD points of the two consecutive subCAEs $K-1$ and K decreasing (see Fig. 5b and c). When this slope does not decrease anymore, we fix $\lambda^{(K-1)}$ and continue the process recursively. The overall effect of the λ scheduling is to progressively accommodate the target RD point for each subCAE so training can approximate the optimal RD points.

5. Experiments

5.1. Experimental settings

We implemented[2] and evaluated the proposed approaches building upon the widely used image compression framework proposed by Balle at al. [6] which typically uses layers with a width of 192 filters (encoder: 3 conv, 3 GDN; decoder: 3 deconv, 3 IGDN). To address different complexities and rates, we consider five different widths ($w \in \{48, 72, 96, 144, 192\}$), which also control the total capacity of the model. The models are trained and evaluated

[2] https://github.com/FireFYF/SlimCAE

Algorithm 1 SlimCAE training with λ-scheduling

```
Input: $\mathcal{X}_{tr}$, $\mathcal{X}_{val}$                     ▷ Training, val. data
Input: SlimCAE with $K$ subCAEs and parameters $\Psi$
Input: $\lambda^{(K)}$                     ▷ RD tradeoff for largest subCAE
Input: $\kappa, T, M$                     ▷ Other hyperparameters
Output: $\Psi, \Lambda$
 1: $\Lambda_0 \leftarrow \left[\lambda^{(K)}, \ldots, \lambda^{(K)}\right]$
 2: $\Psi_0 \leftarrow \arg\min_\Psi \mathcal{L}\left(\Psi; \mathcal{X}_{tr}, \Lambda_0\right)$          ▷ Naive training
 3: Calculate $R_0^{(K-1)}, R_0^{(K)}, D_0^{(K-1)}, D_0^{(K)}$ over $\mathcal{X}_{val}$
 4: $\xi_0 \leftarrow \frac{D_0^{(K)} - D_0^{(K-1)}}{R_0^{(K)} - R_0^{(K-1)}}$
 5: $t \leftarrow 1$
 6: for $i \leftarrow K-1$ to 1 do
 7:     for $m \leftarrow 1$ to $M$ do
 8:         $\Lambda_i \leftarrow \left[\kappa\lambda_{i-1}^{(1)}, \ldots, \kappa\lambda_{i-1}^{(i)}, \lambda_{i-1}^{(i+1)}, \ldots, \lambda_{i-1}^{(K)}\right]$
 9:         for $j \leftarrow 1$ to $T$ do
10:             $\Psi_t \leftarrow \arg\min_\Psi \mathcal{L}\left(\Psi_{t-1}; \mathcal{X}_{tr}, \Lambda_i\right)$
11:             $t \leftarrow t+1$
12:         end for
13:         if $R_t^{(i+1)} < R_t^{(i)}$ then
14:             continue
15:         else
16:             Calculate $\xi_t \leftarrow \frac{D_t^{(i+1)} - D_t^{(i)}}{R_t^{(i+1)} - R_t^{(i)}}$ over $\mathcal{X}_{val}$
17:             if $\xi_t > \xi_{t-T}$ then
18:                 break
19:             end if
20:         end if
21:     end for
22: end for
```

on the CLIC dataset[3]. In Section 5.6 we evaluate on the Kodak [4] and Tecnick[5] datasets to compare to other methods. We evaluate both RD performance and efficiency (memory footprint, computational cost and latency).

SlimCAE variants. We consider three GDN/IGDN variants (SwitchGDN, SlimGDN and SlimGDN+ corresponding to Fig. 3a, b and c respectively) and three training strategies (naive, estimated λs and λ-scheduling, corresponding to the methods described in Sections 3.5, 4.2 and 4.3). We used a switchable entropy model (i.e. one independent entropy model per width), since the number of parameters is negligible compared to the overall model.

Baselines. We compare SlimCAE to independently trained CAEs for different widths (five models in total following [6]). We also compare to three approaches to provide variable rate in a single model: bottleneck scaling (BScale) [36][6], modulated autoencoders (MAE) [42] and conditonal autoencoders (cAE) [10].

Training details. We use 240×240 pixel crops and a batch size of 8. Some methods are trained in one step, while other require two steps. The former includes independent CAEs, MAE, cAE, SlimCAE with naive training and training with estimated λs. In these cases we use a learning rate of 1e-4 (1e-3 for the entropy model) during 1.2M iterations, and then halve them for an additional 200K iterations. SlimCAE with λ-scheduling uses a SlimCAE after naive training 1.2M iteration, followed by training with λ-scheduling ($\kappa = 0.8$, $T = 2000$ and $M = 7$ in Alg. 1)[7] during 28K iterations and then fine tuned (by halving the learning rates) until a total of 1.5M iterations with the final λs fixed. We measure distortion as MSE during training and as PSNR during λ-scheduling. BScale uses a CAE trained during 1.2M iterations, which is then fixed and scaling parameters are learned during 300K iterations.

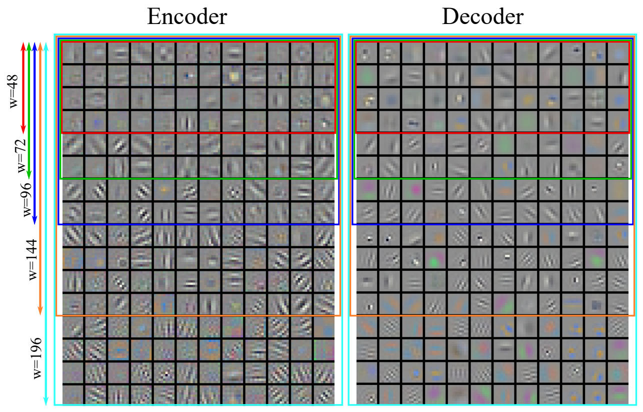


Figure 6: Filters in the first convolutional layer (encoder) and last convolutional layer (decoder) for different widths.

5.2. Qualitative analysis

SlimCAE can effectively distribute and optimize the capacity in a way that each subCAE can focus on the patterns relevant to its own optimal RD tradeoff. For example, the first convolutional layer of the smallest subencoder (see Fig. 6) contains only filters sensitive to low frequency patterns, while larger subencoders progressively include filters related with higher frequency, since they are necessary to achieve lower distortion. The latent representation in the bottleneck is also structured in a similar way from coarse reconstruction to fine details (see Fig. 11 a-b in supp. mat.).

5.3. Rate-distortion

Fig. 7a shows the rate-distortion performance obtained with different GDN variants. Sharing GDN parameters

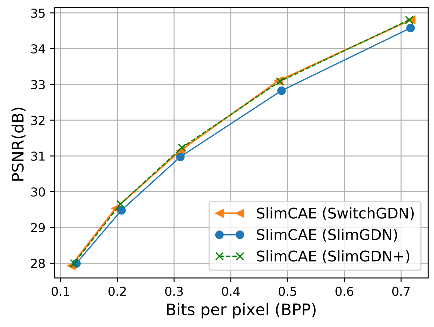
[3] https://www.compression.cc/2019/challenge

[4] http://r0k.us/graphics/kodak

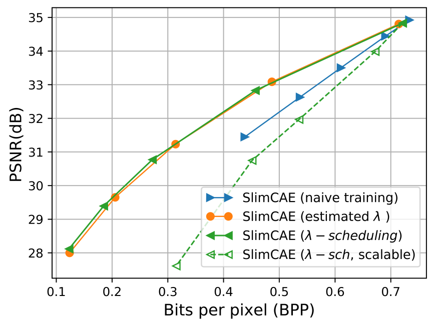
[5] https://testimages.org/sampling

[6] Note that [36] actually introduces the term *compressive autoencoder*. s in a general sense, and in our experiments CAE refers to our baseline [6], while BScale denotes the variable rate approach in [36]

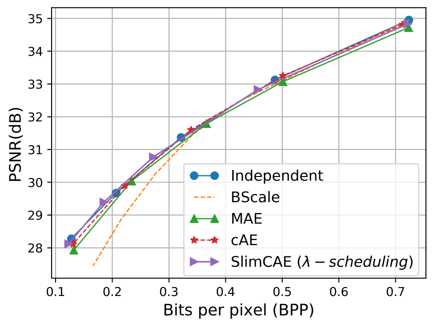
[7] We extend the implementation of [6], which optimizes $\lambda D + R$. We adapt λ-scheduling correspondingly.



(a) GDN variants (with estimated λs)



(b) Training strategies and scalability



(c) Variable rate methods

Figure 7: Rate-distortion performance comparison (CLIC dataset).

across different widths (i.e. SlimGDN) results in worse performance than independent ones (i.e. SwitchGDN). However, this loss can be recovered when parameter modulation (i.e. SlimGDN+) at a negligible parameter increase.

Fig. 7b shows that naive training suffers from the limitations of using a single shared tradeoff λ, while training with more adequate width-specific λs (those estimated in Fig. 4) results in an RD curve closer to the obtained with independent CAEs. Fig. 7b also illustrates the effect of λ-scheduling in the RD curve, which gets progressively closer until it essentially achieves the same performance as independent CAEs (see Fig. 10 in supplementary for more details), but without requiring training auxiliary models.

Finally, we compare SlimCAE to other baselines enabling variable rate in a single model. SlimCAE obtains the best RD performance, overlapping with that of independent CAEs, but with a much lower training and memory cost, as we see next.

5.4. Efficiency

We also evaluate the efficiency of SlimCAE in terms of memory footprint (in MB), computational cost (in FLOPs) and latency (in ms)[8]. Values in features and parameters are represented with 4 bytes, and the features are calculated for input images of size 768 $\times$ 512 pixels. We consider a baseline with five independent CAEs optimized for different RD tradeoffs. For fair comparison we use the minimum width that guarantees that the RD performance at a particular tradeoff λ remains optimal (see Fig. 4).

Fig. 8 shows the memory footprint in the encoder and decoder at different widths. Features require significantly more memory than model parameters, especially at small widths. While for the largest width all methods require similar memory, an independent CAE and SlimCAE can reduce significantly the memory footprint for small widths. This reduction also results in a significantly lower computational cost (see Table 2), and much lower latency during both encoding and decoding (see Table 3). Now we consider the total memory required to provide the five different rates. It requires to store the model parameters of the independent CAEs (31.1 MB), in contrast to just a single model

Table 2: Computational cost of trained models (millions of FLOPs). Some methods adjust layer widths.

Methods	Low rate	→	Medium rate	→	High rate
Independent	**15.34M** (w=48)	**31.69M** (w=72)	**53.81M** (w=96)	**115.53M** (w=144)	**200.28M** (w=192)
BScale [36]	**200.28M** (w=192)	**200.28M** (w=192)	**200.28M** (w=192)	**200.28M** (w=192)	**200.28M** (w=192)
MAE [42]	**200.40M** (w=192)	**200.40M** (w=192)	**200.40M** (w=192)	**200.40M** (w=192)	**200.40M** (w=192)
cAE [10]	**200.31M** (w=192)	**200.31M** (w=192)	**200.31M** (w=192)	**200.31M** (w=192)	**200.31M** (w=192)
SlimCAE	**15.34M** (w=48)	**31.69M** (w=72)	**53.81M** (w=96)	**115.53M** (w=144)	**200.28M** (w=192)

Table 3: Encoding and decoding latency (ms) for a 768×512 input image (i.e. batch size 1) on a NVIDIA GTX 1080Ti GPU (excluding data loading/writing and arithmetic coding.

	Methods	Low rate	→	Medium rate	→	High rate
Encoding	Independent	1.9 ± 0.19	2.2 ± 0.17	2.8 ± 0.22	4.0 ± 0.19	5.1 ± 0.20
	BScale [36]	5.2 ± 0.11	5.2 ± 0.16	5.2 ± 0.22	5.2 ± 0.15	5.2 ± 0.13
	MAE [42]	5.4 ± 0.20	5.4 ± 0.20	5.4 ± 0.13	5.4 ± 0.10	5.4 ± 0.11
	cAE [10]	5.5 ± 0.18	5.5 ± 0.11	5.5 ± 0.14	5.5 ± 0.21	5.5 ± 0.28
	SlimCAE	$\mathbf{1.9 \pm 0.15}$	$\mathbf{2.2 \pm 0.27}$	$\mathbf{2.8 \pm 0.12}$	$\mathbf{4.0 \pm 0.17}$	$\mathbf{5.1 \pm 0.10}$
Decoding	Independent	2.9 ± 0.20	3.5 ± 0.10	4.3 ± 0.07	6.1 ± 0.07	8.0 ± 0.13
	BScale [36]	8.0 ± 0.21	8.0 ± 0.13	8.0 ± 0.18	8.0 ± 0.11	8.0 ± 0.13
	MAE [42]	8.4 ± 0.15	8.4 ± 0.13	8.4 ± 0.08	8.4 ± 0.07	8.4 ± 0.14
	cAE [10]	8.5 ± 0.20	8.5 ± 0.07	8.5 ± 0.09	8.5 ± 0.13	8.5 ± 0.21
	SlimCAE	$\mathbf{2.9 \pm 0.10}$	$\mathbf{3.5 \pm 0.07}$	$\mathbf{4.3 \pm 0.10}$	$\mathbf{6.1 \pm 0.16}$	$\mathbf{8.0 \pm 0.13}$

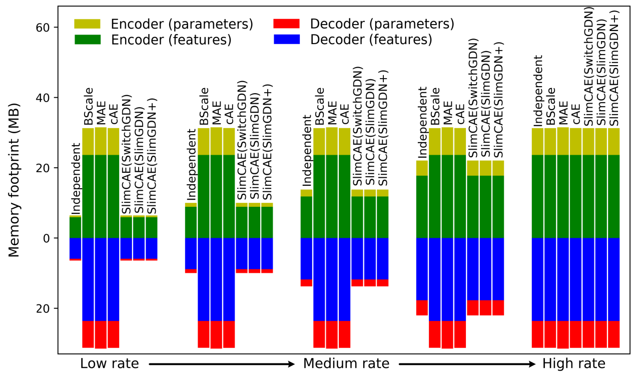


Figure 8: Memory footprint comparison for different rates.

[8]We only compare to NIC codecs, since traditional codecs run in different hardware (CPU instead of GPU). As reference, our BPG baseline takes on CPU (for 0.1-1.0 bpp) 3.2-4.5 s/img (enc) and 95-131 ms/img (dec).

Table 4: BD-rate (%) over BPG. Lower means better.

Dataset	PSNR (opt. for MSE)				MS-SSIM (opt. for MS-SSIM)			
	[7]	[23]	Slim[7]	Slim[23]	[7]	[23]	Slim[7]	Slim[23]
Kodak	9.68	-8.94	9.52	-6.17	-41.46	-46.92	-41.41	-47.88
Tecnick	2.95	-11.77	5.23	-10.43	-41.50	-43.24	-40.34	-48.94

for BScale (15.3 MB), MAE (15.7 MB), cAE (15.3 MB) and SlimCAE (15.3 MB with SlimGDN+). If we consider the memory used to store features (note that at only one model is use at a time), the memory footprint of multiple CAEs varies from 42.9 to 78.3 MB, depending on the selected rate, and similarly to SlimCAE (27.1 to 62.5 MB). In contrast, other methods cannot adapt the complexity and remain with a higher and constant footprint (BScale 62.5 MB, MAE 63 MB and cAE 62.6 MB)[9]. The SlimGDN+ layers require 0.85 MB (compared to 1.71 MB and 0.85 MB in SwitchGDN and SlimGDN, respectively).

Finally, Table 1 summarizes the main advantages and drawbacks of different methods. SlimCAE is the most complete of them providing variable rate with a single model and controllable memory and computational requirements, while achieving optimal RD performance. Training and switching between multiple CAEs suffers from a higher memory footprint (that increases with the number of target RD points), and the much higher cost of training multiple models. The other baselines can adapt rate, but not memory and computational costs, which remain high at low rates.

5.5. Scalable bitstreams

Motivated by SlimCAE's structured latent respresentation, we consider a variant where each group of channels are encoded independently, allowing *quality scalability* and progressive decoding. The resulting bitstreams (i.e. base[+ enhancement stream(s)]) are all decodable by the SlimCAE decoder. On the other hand, the SlimCAE is no longer slimmable, so enabling quality scalability disables memory and computation scalability since the SlimCAE is no longer slimmable, and also has certain penalty in RD performance (see Fig. 7b and Fig. 11 c in supp. mat), a usual compromise in scalable image and video coding [35, 26, 30].

5.6. Slimmable entropy models

Our approach is general and can be easily extended including slimmable versions of entropy models to achieve state-of-the-art RD performance. Following [7, 22], we train[10] a larger capacity autoencoder[11] with a three conv layer slimmable hyperprior[12] [7] and conditional convolutions [10]. We also include a slimmable autoregressive context model[13] [22]. We trained these models[14] and evaluated on Kodak and Tecnik datasets. Fig. 9 shows that our approach can be integrated with almost no penalty in RD performance, while providing the aforementioned advantages in terms of rate and complexity control of SlimCAEs. The same conclusions hold when optimizing MS-SSIM, keeping a significant gain over BPG (see Table 4).

[9]Note that, in practice, an optimized implementation could save some memory by discarding intermediate features once they are processed.

[10] 3M iter., batch 8, 256 $\times$ 256 crops, λ-sched ($\kappa = 0.8, T = 10^4, M = 7$).

[11]$w \in \{96, 144, 192, 288, 384\}$

[12]$w \in \{48, 72, 96, 144, 192\}$, and Leaky ReLUs (same in decoder)

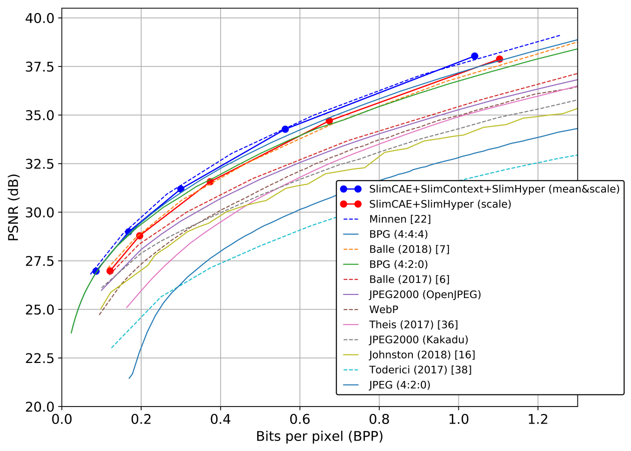


Figure 9: Rate-distortion performance of SlimCAE with slimmable entropy model (Kodak dataset).

6. Conclusion

Neural image compression is a new paradigm for image (and by extension video) coding, with numerous advantages over the traditional handcrafted linear transform coding. However, current approaches are also resource demanding, and usually tied to a particular rate, which limits their application in practice.

Our approach is thus a step further towards practical and adaptive learned image compression, combining in a single model important functionalities, such as excellent rate-distortion performance, low and dynamically adjustable memory footprint, computational cost and latency, all of them easily controlled via a lightweight switching mechanism. This makes our approach attractive to resource-limited devices (e.g. smartphones), when rate and computation needs to be controlled dynamically (e.g. video coding, multi-tasking) or to deploy different models to heterogeneous devices, adapted to their computational capabilities. SlimCAE can also generate scalable bitstreams, which can be useful in streaming and broadcasting scenarios with heterogeneous devices. In this paper we also study the fundamental connection between rate-distortion performance and network capacity, and propose an efficient and effective approach to train the slimmable model in a single pass.

[13]One masked conv layer with $w \in \{96, 144, 192, 288, 384\}$

[14]On CLIC extended with 20k high quality images from flickr.com

7. Acknowledgments

We acknowledge the support from Huawei Kirin Solution and the Spanish Government funding for projects RTI2018-102285-A-I00 and RYC2019-027020-I.

References

[1] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems*, pages 1141–1151, 2017. 2

[2] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 221–231, 2019. 2

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. 2

[4] Johannes Ballé, Valero Laparra, and Eero Simoncelli. Density modeling of images using a generalized normalization transformation. In *4th International Conference on Learning Representations, ICLR 2016*, 2016. 4

[5] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. Density modeling of images using a generalized normalization transformation. *arXiv preprint arXiv:1511.06281*, 2015. 2

[6] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016. 1, 2, 3, 4, 5, 6

[7] 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. 2, 4, 8

[8] Chunlei Cai, Li Chen, Xiaoyun Zhang, and Zhiyong Gao. Efficient variable rate image compression with multi-scale decomposition network. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018. 2

[9] Chunlei Cai, Li Chen, Xiaoyun Zhang, Guo Lu, and Zhiyong Gao. A novel deep progressive image compression framework. In *2019 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2019. 2

[10] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Variable rate deep image compression with a conditional autoencoder. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3146–3154, 2019. 1, 2, 3, 6, 7, 8

[11] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In *Advances In Neural Information Processing Systems*, pages 3549–3557, 2016. 1

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[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. 2

[14] Nick Johnston, Elad Eban, Ariel Gordon, and Johannes Ballé. Computationally efficient neural image compression. *arXiv preprint arXiv:1912.08771*, 2019. 2

[15] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4385–4393, 2018. 1

[16] AH Khan and EL Hines. Integer-weight neural nets. *Electronics Letters*, 30(15):1237–1238, 1994. 2

[17] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack. Context-adaptive entropy model for end-to-end optimized image compression. In *International Conference on Learning Representations*, 2018. 2

[18] Mu Li, Kede Ma, Jane You, David Zhang, and Wangmeng Zuo. Efficient and effective context-based convolutional entropy modeling for image compression. *IEEE Transactions on Image Processing*, 29:5900–5911, 2020. 2

[19] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression. In *conference on Computer Vision and Pattern Recognition*, pages 3214–3223, 2018. 1

[20] Dong Liu, Haichuan Ma, Zhiwei Xiong, and Feng Wu. Cnn-based dct-like transform for image compression. In *International Conference on Multimedia Modeling*, pages 61–72. Springer, 2018. 1

[21] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402, 2018. 1, 2

[22] 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. 1, 2, 4, 8

[23] David Minnen and Saurabh Singh. Channel-wise autoregressive entropy models for learned image compression. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3339–3343. IEEE, 2020. 2

[24] Ken M Nakanishi, Shin-ichi Maeda, Takeru Miyato, and Daisuke Okanohara. Neural multi-scale image compression. In *Asian Conference on Computer Vision*, pages 718–732. Springer, 2018. 2

[25] Majid Rabbani. Jpeg2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging*, 11(2):286, 2002. 1

[26] Hayder M Radha, Mihaela Van der Schaar, and Yingwei Chen. The mpeg-4 fine-grained scalable video coding method for multimedia streaming over ip. *IEEE Transactions on multimedia*, 3(1):53–68, 2001. 8

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 2

[28] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *International Conference on Machine Learning*, pages 2922–2930, 2017. 2

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2

[30] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Overview of the scalable video coding extension of the h. 264/avc standard. *IEEE Transactions on circuits and systems for video technology*, 17(9):1103–1120, 2007. 8

[31] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001. 1

[32] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012. 1

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[34] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 2

[35] David Taubman. High performance scalable image compression with ebcot. *IEEE Transactions on image processing*, 9(7):1158–1170, 2000. 8

[36] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017. 1, 2, 3, 6, 7

[37] George Toderici, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*, 2015. 1, 2

[38] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017. 1

[39] Michael Tschannen, Eirikur Agustsson, and Mario Lucic. Deep generative models for distribution-preserving lossy compression. In *Advances in Neural Information Processing Systems*, pages 5929–5940, 2018. 2

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

[41] Thomas Wiegand and Heiko Schwarz. *Source coding: Part I of fundamentals of source and video coding*. Now Publishers Inc, 2011. 1

[42] Fei Yang, Luis Herranz, Joost van de Weijer, José A Iglesias Guitián, Antonio M López, and Mikhail G Mozerov. Variable rate deep image compression with modulated autoencoder. *IEEE Signal Processing Letters*, 27:331–335, 2020. 1, 2, 3, 6, 7

[43] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1803–1811, 2019. 2

[44] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *7th International Conference on Learning Representations, ICLR 2019*, 2019. 2, 3, 4, 5

[45] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017*, 2016. 2

A. Example of λ-scheduling

Fig. 10 (top) illustrates the evolution of the RD curve from the initial naive training (in black) until the end of λ-scheduling (in red). Fig. 10 (bottom) illustrates the schedule for the λs of the different subCAEs.

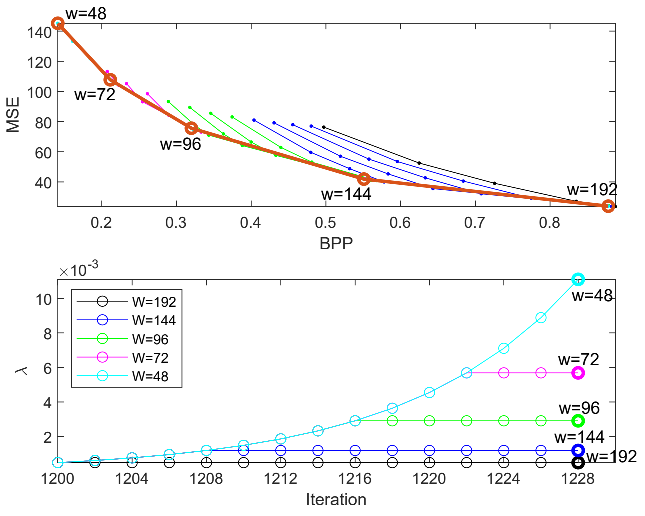


Figure 10: Evolution of the RD curve (top) and λ during the λ-scheduling phase. Naive training shown in black (top).

B. Additional visualizations

Fig. 11a shows an illustrative example of the reconstructed images and (quantized) latent representations obtained by the different subCAEs. The slimmable structure of the last layer of the encoder results in a latent representation also structured in five groups of channels (i.e. 1-48, 49-72, 73-96, 97-144 and 145-192). One representative channel per group is shown. The corresponding bit allocation is shown in Fig. 11b. Note that smaller subCAEs also allocate fewer bits in the first group of channels.

Fig. 11c illustrates progressive decoding when (quality) scalability is enabled, compared with the default slimmable decoding (non scalable bitstreams). In this case each group of channels of $w = 192$ in Fig. 11b correspond to the base stream (channels 1-48) and four enhancement streams, which can be progressively combined to improve quality. However, there is a noticeable increase in both rate (explained by the bit allocation shown in Fig. 11b) and distortion with the quality scalable bitstream.

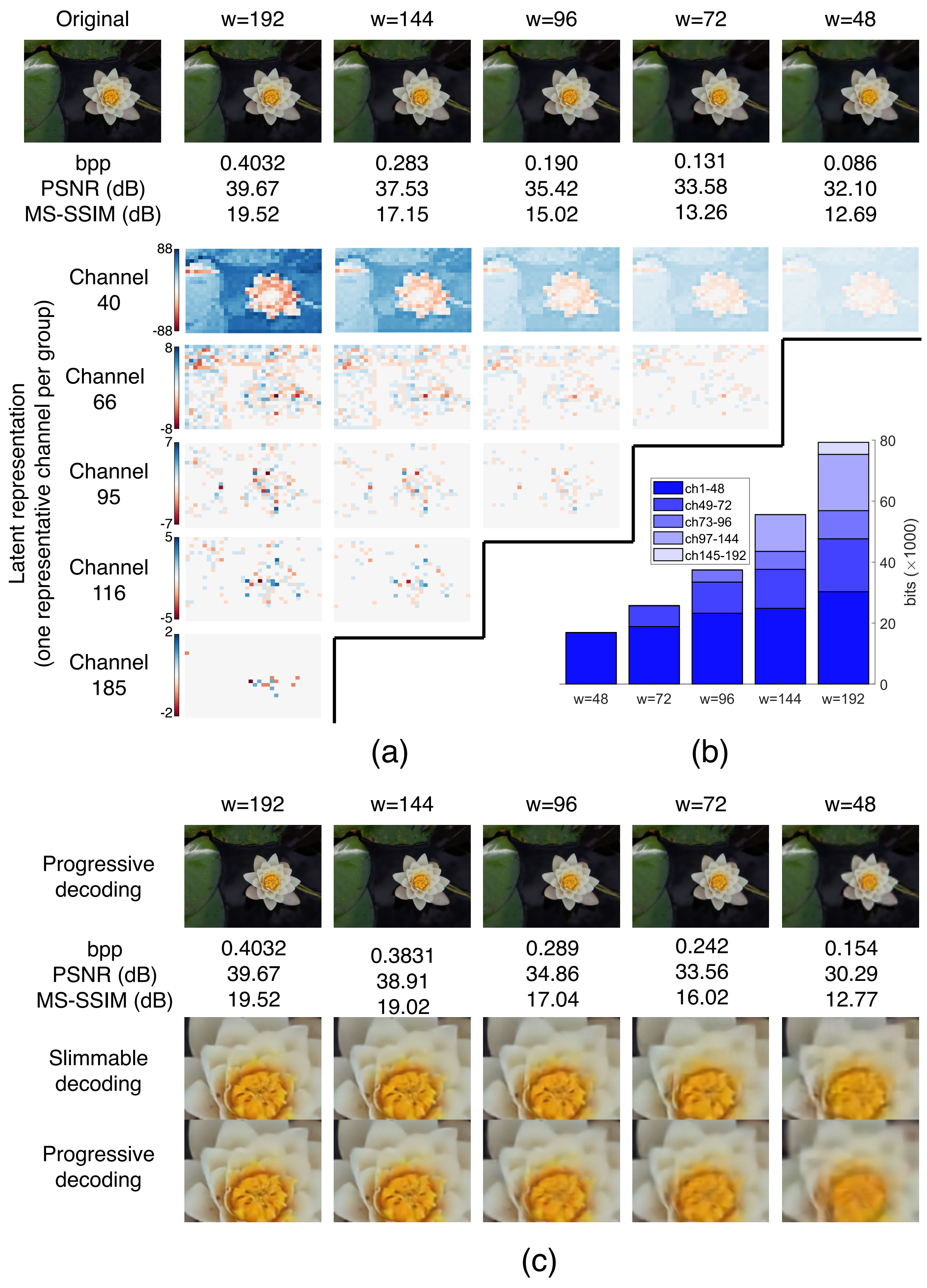


Figure 11: Illustrative example: (a, top) reconstructions using slimmable decoding, (a, bottom) selection of quantized latent maps (one per group of channels), (b) break down of bits spent in each group of channels, (c) scalable bitstream.