Lossless Image Compression Using a Multi-Scale Progressive Statistical Model

Honglei Zhang[1] [0000−0002−8229−852X], Francesco Cricri[1] [0000−0002−1521−420X],
Hamed R. Tavakoli[1] [0000−0002−9466−9148], Nannan Zou[1,2] [0000−0001−5553−5975],
Emre Aksu[1] [0000−0001−7363−2824], and Miska M.
Hannuksela[1] [0000−0003−3405−0850]

[1] Nokia Technologies, Tampere, Finland
[2] Tampere University, Finland
{honglei.1.zhang, francesco.cricri, hamed.rezazadegan_tavakoli,
nannan.zou.ext, emre.aksu, miska.hannuksela}@nokia.com
http://www.nokia.com

Abstract. Lossless image compression is an important technique for image storage and transmission when information loss is not allowed. With the fast development of deep learning techniques, deep neural networks have been used in this field to achieve a higher compression rate. Methods based on pixel-wise autoregressive statistical models have shown good performance. However, the sequential processing way prevents these methods to be used in practice. Recently, multi-scale autoregressive models have been proposed to address this limitation. Multi-scale approaches can use parallel computing systems efficiently and build practical systems. Nevertheless, these approaches sacrifice compression performance in exchange for speed. In this paper, we propose a multi-scale progressive statistical model that takes advantage of the pixel-wise approach and the multi-scale approach. We developed a flexible mechanism where the processing order of the pixels can be adjusted easily. Our proposed method outperforms the state-of-the-art lossless image compression methods on two large benchmark datasets by a significant margin without degrading the inference speed dramatically.

1 Introduction

A lossless image compression system converts an input image into a bitstream that is smaller in size and the original image can be fully reconstructed. These systems bring several benefits in data storage, manipulating, and transferring where information loss is not allowed. On the other hand, lossy image compression aims at a much higher compression rate by allowing the data to be reconstructed with a certain amount of degradation. PNG [1] and FLIF [2] are image formats that support lossless image compression. Other image compression techniques, such as JPEG 2000 [3], WebP[4], and HEVC [5] support both lossy and lossless image compression.

A typical lossless image compression system contains a probability model and an entropy codec. The probability model is built up from a statistical model that gives an estimation of the statistical characteristics of an input image. The entropy codec encodes/decodes symbols in the input data into/from variable-length prefix-free codes according to the statistics given by the probability model. According to Shannon′s source coding theorem [6], the optimal code length of a symbol is $-\log_2 P$, where P is the probability of the symbol to be encoded. Arithmetic coding is an efficient entropy coding algorithm that has been adopted in many image/video coding systems, such as HEVC [5].

Accurate probability estimation is the most critical aspect of a lossless image compression system. Traditional lossless image compression system, such as PNG [1] uses a static statistical model with Huffman coding to encode the difference of the pixels and the predicted value based on its neighboring pixels. AVC [7] and HEVC [5] applies an adaptive model, known as CABAC, to give a better estimation of the probabilities [8]. The CABAC model updates the model parameters during the process of encoding/decoding according to the content of the image. With the fast development of deep learning techniques, deep neural networks have been used to achieve more accurate statistical estimations [9–15]. Unlike traditional image/video coding systems [5, 7], which encode residuals between the input symbols and the predicted symbols, deep neural network-based image/video compression systems normally use the estimated value distribution function directly, since deep neural networks are more capable of modeling complicated distribution functions.

An autoregressive statistical model uses pixels that have already been processed as a context to derive the value distribution function of the pixels to be encoded/decoded [16–18]. These models can achieve good compression rates. However, at the inference stage, i.e. encode/decoding stage, the input data are processed sequentially and an expensive deep neural network is involved to process every pixel. Thus, the systems based on these models are only capable of encoding/decoding small images [15]. To address this problem, L3C [15] and SReC [14] adopt a multi-scale approach. Pixels at each scale are processed in a parallel manner, which greatly speeds up the encoding/decoding procedure. However, the compression rate is compromised significantly. In [13], the authors proposed another type of context-based statistical model where an input image is first compressed with a lossy compression method and then used as the context to infer the value distribution function of the pixels in an input image.

Flow-based approach [9, 12, 19] learns a invertible function that converts an input image into a latent representation with a predefined distribution function. At the decoding stage, the inverse function of the encoding function is applied to reconstruct the input image. These algorithms also process pixels in a parallel manner and have similar computation complexities for encoding and decoding. However, the capacity of these methods is limited by the constrain that the encoding and decoding functions must be mutually invertible. Large neural networks have to be used to improve the performance of the system.

In this paper, we propose a lossless image compression system based on a multi-scale progressive statistical model. The proposed method significantly improves the compression rate without heavily degrading the encoding/decoding speed. Our system incorporates a flexible framework where the compression rate and encoding/decoding speed can be adjusted easily. Using the proposed model, we have improved the state-of-the-art compression rate on the ImageNet64 dataset and the OpenImage dataset by a significant margin.

2 Related Work

2.1 Pixel-wise autoregressive statistical model

Pixel-wise autoregressive statistical models, such as PixelRNN [17], PixelCNN [16] and PixelCNN++ [18], model the joint distribution of the pixels in an image as the product of the distribution of each pixel conditioned on the previous pixels. Let x be an image, N be the number of pixels, and $x_1, x_2, \cdots, x_N$ be the pixels arranged in a certain order, for example, a raster scan order. The joint distribution of x is defined by

$$p(x) = \prod_{i=1}^{N} p\left(x_i | x_1, x_2, \cdots, x_{i-1}\right), \tag{1}$$

where $p\left(x_i | x_1, x_2, \cdots, x_{i-1}\right)$ is modeled by a deep neural network. During the training and encoding stage, a masked convolutional neural network (CNN) is applied to ensure that the probability estimation of the current pixel only depends on the previous pixels. More importantly, with the masked CNN architecture the system can be trained efficiently as a standard deep CNN. However, at the decoding stage, pixels can only be processed sequentially. This limitation prevents the pixel-wise autoregressive model from being used in practice [15].

2.2 Multi-scale autoregressive statistical model

Multi-scale autoregressive statistical models proposed in L3C [15] and SReC [14] model the joint distribution of x as the product of the conditional distributions in multiple scales. Let M be the number of scales, and $x^{(i)}$ be the representation at scale i. For simplicity, we use $x^{(0)}$ to denote the input image. We have $x^{(i)} = E\left(x^{(i-1)}\right)$, where $E(\cdot)$ is a encoding function that performs a type of downsampling operation to an input representation. Let $C^{(i)} = \left\{x^{(i)}, x^{(i+1)}, \cdots, x^{(M)}\right\}$ be the set of representations from scale i to scale M. The joint distribution of $x^{(0)}, x^{(1)}, \cdots, x^{(M)}$ is defined by

$$p\left(x^{(0)}, x^{(1)}, \cdots, x^{(M)}\right) = \left(\prod_{i=0}^{M-1} p\left(x^{(i)} | C^{(i+1)}\right)\right) p\left(x^{(M)}\right), \tag{2}$$

where $p\left(x^{(M)}\right)$ is the distribution function of the last scale and modeled by the assumption that the elements in $x^{(M)}$ are independent and uniform distributed. The conditional distribution at scale i is parameterized as

$$p\left(x^{(i)}|C^{(i+1)}\right) = p\left(x^{(i)}|y^{(i+1)}\right), \tag{3}$$

where $y^{(i)} = D\left(y^{(i+1)}, x^{(i)}\right)$, and $D(\cdot)$ is a decoder function implemented using a deep neural network. The system is trained to optimize the cross entropy with respect to the estimated probability distribution $p\left(x^{(0)}|y^{(1)}\right)$.

2.3 Flow-based statistical model

Flow-based generative method is another technique that has been recently used for lossless image compression, for example, in LBB [9], Integer Flows [12], and Real-NVP [19]. A flow-based method learns an invertible mapping function from the image space to a latent space where the dependent variable follows a predefined distribution function. Let the mapping function be $z = f(x)$ where z is the dependent variable in the latent space with a predefined distribution function $p_Z(z)$. The distribution function of input x is defined by

$$p(x) = p_Z(f(x))\left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right|, \tag{4}$$

where $\det(\cdot)$ is the determinant of a matrix. Because of the determinant operation, the Jacobian of $f(x)$ at x must have certain forms, for example, be a diagonal or upper triangular matrix. In Real-NVP [19], Dinh et.al. proposed a invertible function using coupling layers that are defined by

$$z_{1:d} = x_{1:d} \tag{5}$$

$$z_{d+1:N} = x_{d+1:N} \odot \exp\left(s\left(x_{1:d}\right)\right) + t\left(x_{1:d}\right), \tag{6}$$

where N is the dimension of the input variable, $s(\cdot)$ and $t(\cdot)$ are scale and translation functions, and $\odot$ is element-wise product. The coupling layer partition the variables in x into two sets $x_{1:d}$ and $x_{d+1:N}$. $x_{1:d}$ is directly mapped to the corresponding output variables. $x_{d+1:N}$ is mapped to the output variables using scale and translation factors that are derived from $x_{1:d}$.

Next, we will give a detailed description of our proposed statistical model and our lossless image compression system.

3 Multi-Scale Progressive Statistical Model

3.1 Statistical model

Similar to multi-scale approaches [14, 15], we downsample an input image into a number of low-resolution representations. Pixels in lower resolution representations are used as a context to estimate the distribution function of the pixels in a higher resolution representation.

In L3C [15], a representation $x^{(i+1)}$ is derived from representation $x^{(i)}$ using an encoder function $E(\cdot)$. This design not only complicates the system but also makes the training less efficient since the system becomes a very deep neural network from the end-to-end point of view. Experiments show that increasing the number of scales does not improve the performance [15]. In SReC [14], the encoder function is defined as an average pooling function. SReC can be trained more efficiently than L3C. However, extra bits must be used to encode round errors. Inspired by the coupling layer in Real-NVP [19], we simply take a subset of pixels in $x^{(i)}$ as $x^{(i+1)}$. With this simplification, the probability distribution function $p\left(x^{(0)}\right)$ can be directly factorized as

$$p\left(x^{(0)}\right)=\left(\prod_{i=0}^{M-1} p\left(x^{(i)}|C^{(i+1)}\right)\right)p\left(x^{(M)}\right) \tag{7}$$

To further improve the compression rate without dramatically compromizing the encoding/decoding speed, we partition the pixels at each scale into groups, i.e. $x^{(i)}=g_1^{(i)}\cup g_2^{(i)}\cup\cdots\cup g_{B_i}^{(i)}$, where B_i is the number of groups at scale i. The details of the grouping methods will be described in Section 3.3. The groups are processed sequentially. Once all pixels in a group are processed, they are added to the context to improve the estimation accuracy of the pixels in the next group. Taking this grouping operation into consideration, we can further factorize Eq. 7 to

$$p\left(x^{(i)}|C^{(i+1)}\right)=\prod_{j=1}^{B_i} p\left(g_j^{(i)}|g_1^{(i)},g_2^{(i)},\cdots,g_{j-1}^{(i)},C^{(i+1)}\right), \tag{8}$$

where $g_j^{(i)}$ is the pixel group j at scale i, B_i is the number of groups at scale i, and $i=0,1,\cdots,M-1$. Let $G_j^{(i)}=\left\{g_1^{(i)},g_2^{(i)},\cdots,g_{j-1}^{(i)},C^{(i+1)}\right\}$ be the context for group $g_j^{(i)}$. Assuming the pixels in a group are conditional independent, we have the probability distribution function

$$p\left(g_j^{(i)}|G_j^{(i)}\right)=\prod_{k=1}^{N_j^{(i)}} p\left(x_{j,k}^{(i)}|G_j^{(i)}\right), \tag{9}$$

where $x_{j,k}^{(i)}\in g_j^{(i)}$ is pixel k in group j at scale i, and $N_j^{(i)}$ is the number of pixels in group $g_j^{(i)}$.

Next, we use a mixture of logistic distributions to model $p\left(x_{j,k}^{(i)}|G_j^{(i)}\right)$ in a similar way as defined in PixelCNN++ [18]. The parameters of the mixture model $p\left(x_{j,k}^{(i)}|G_j^{(i)}\right)$ are derived from a function of context $G_j^{(i)}$ using a deep neural network. Note that, in this model, the logistic mean of the distribution function of a subpixel is modeled as a weighted sum of the logistic means of previous subpixels.

Given the statistical model defined by Eqs. 7, 8, and 9, the deep neural network is trained to minimize the cross entropy of the input $x^{(0)}$, i.e.

$$\mathbb{E}_{x^{(0)}\sim q(x)}\left[-\log p\left(x^{(0)}\right)\right], \tag{10}$$

where $q(x)$ is the true distribution of the input image and $\mathbb{E}(\cdot)$ is the expectation of a random variable.

3.2 Lossless image compression system

Figure 1 illustrates our lossless image compression system using the proposed multi-scale progressive statistical model. For clarity, the figure only shows a system of two scales.

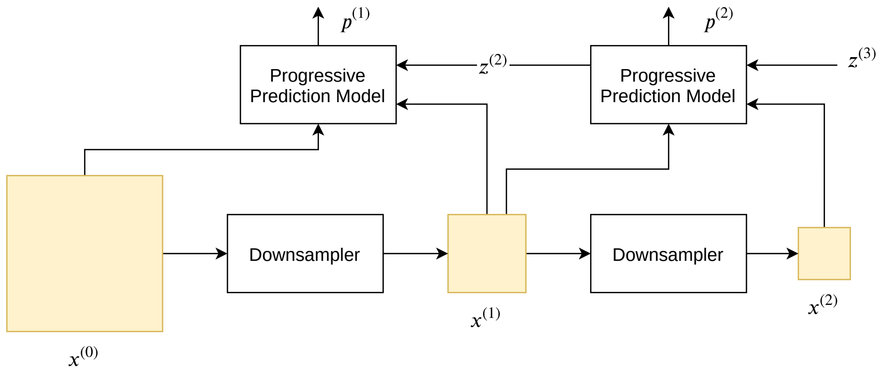

Fig. 1. The lossless image compression system using the proposed multi-scale progressive statistical model. Yellow boxes indicate tensors.

In this system, an input image $x^{(0)}$ is first downsampled to low-resolution representations $x^{(1)}$ and $x^{(2)}$. As mentioned in Section 3.1, we take a subset of pixels in $x^{(i)}$ as $x^{(i+1)}$. If the pixels are selected in a checkerboard pattern, the system is equivalent to using the nearest neighbor downsampling operation as the encoder function $E(\cdot)$ as used in [14, 15]. This simplification improves training efficiency and helps us to develop a more flexible architecture to achieve better performance. With this design, we can increase the number of scales without suffering from the gradient vanishing/exploding problem that a very deep neural network system may encounter.

Figure 2 illustrates the progressive prediction model at scale i, which is the core component of our system. An input tensor $x^{(i)}$ is first upsampled to $\hat{x}_1^{(i)}$ to have the same size as tensor $x^{(i-1)}$. For simplicity, we choose the nearest-neighbor upsampling method in our experiments. Next, $\hat{x}_1^{(i)}$ is partitioned into multiple groups as described in Section 3.1. For each group of pixels, a deep CNN is used to estimate the parameters for the value distribution functions of the pixels in that group. The detailed structure of this deep CNN is illustrated on the right side of Figure 2. The deep CNN takes three inputs: $\hat{x}_j^{(i)}$ is the tensor that holds a mixture of predicted and true values of pixels in $x^{(i-1)}$; $m_j^{(i)}$ is the binary mask that indicates the true values in $x_j^{(i)}$; $z_{j-1}^{(i)}$ is the context information

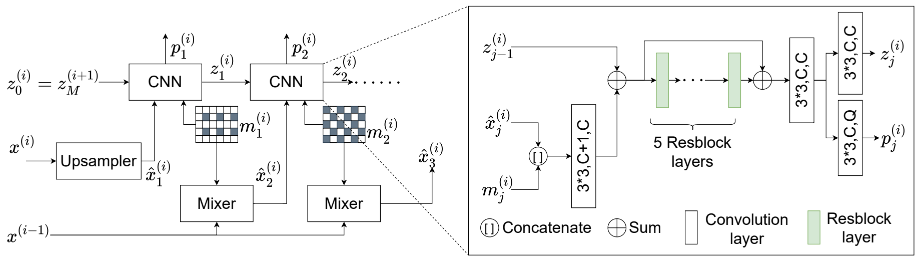

Fig. 2. The architecture of the progressive prediction model in Figure 1. The notations in convolution blocks are in the format of "kernel_size*kernel_size,input_channels,output_channels". The Resblock layer is the basic Resnet block defined in [20]. No batch normalization is used in the system. C is the network size and Q is the number of parameters for the mixture of logistic distributions model.

passed from the previous stage. The initial context $z_0^{(i)}$ is set to be the output context of scale $i+1$. For the last scale when there is no context information available, we set the context to be zero, i.e. $z_0^M = 0$. The deep CNN outputs parameters $p_j^{(i)}$ for the estimated value distribution functions of the pixels in group j, and the context information $z_j^{(i)}$.

At the training and encoding stage, where the ground truth $x^{(i-1)}$ is available, after the pixels in group j have been processed, the system sets the corresponding values in $\hat{x}_{j+1}^{(i)}$with the true pixel values using the mixer component, and updates mask $m_{j+1}^{(i)}$ accordingly. Then, $\hat{x}_{j+1}^{(i)}$, $m_{j+1}^{(i)}$ and $z_j^{(i)}$ are passed to the deep CNN to derive $p_{j+1}^{(i)}$ and $z_{j+1}^{(i)}$. The entropy of the pixels in group $j+1$ is calculated using $p_{j+1}^{(i)}$ and $x^{(i)}$ at the training stage. At the encoding stage, the pixels are encoded by the arithmetic encoder according to the estimated value distribution function using parameters $p_{j+1}^{(i)}$. At the decoding stage, the estimated value distribution functions for pixels in group j are used to decode the true values from the bitstream using the arithmetic decoder. $\hat{x}_{j+1}^{(i)}$ and $m_{j+1}^{(i)}$are updated in the same way as the training and encoding stage. This procedure continues until all $B^{(i)}$ groups are processed.

3.3 Pixel grouping methods

Using the binary mask $m_j^{(i)}$, one can easily set the number of groups and the method to group the pixels. The grouping methods determine the order of the pixels to be processed. In an extreme case when $B^{(i)}$ is equal to the number of pixels to be processed at scale i, the statistical model is equivalent to a pixel-wise autoregressive model described in Section 2.1. Next, we show some grouping methods that can be applied in the proposed system.

Random grouping In this method, pixels to be processed at scale i are randomly assigned to $B^{(i)}$ groups. Note that the grouping is performed in advance and agreed between the encoder and decoder so that the pixels can be decoded correctly at the decoding stage.

Grouping with a fixed pattern With this method, the pixels are grouped by a predetermined fixed pattern. Figure 3 shows two options for the fixed pattern grouping. In (a), an input image is first partitioned into 4x4 blocks and the pixels are grouped according to the index number shown in the figure. Note that this grouping method is similar to the architecture in [14]. Since the value distribution function are estimated using the available pixels in the neighboring area as the context, it is intuitive to process the pixels that have the most available pixels in its neighboring area. In (b), the pixels are assigned to 6 groups in the order of the number of available neighboring pixels.

1	2	1	2	1	2
3	x	3	x	3	x
1	2	1	2	1	2
3	x	3	x	3	x

(a)

5	3	2	4	5	3	2	4
6	x	1	x	6	x	1	x
5	3	2	4	5	3	2	4
6	x	1	x	6	x	1	x

(b)

Fig. 3. Two grouping methods with fixed patterns. (a) Grouping method (3 groups) by a fixed pattern similar to [14]; (b) Grouping method (6 groups) according to the number of availability pixels in the neighboring area. "x" indicates available pixels before grouping. The number indicates the group to which the pixel belongs. Pixels are processed group by group according to their group index number.

Dynamic grouping Instead of using a predefined fixed pattern, one can also group the pixels dynamically according to the content of the image. For example, the grouping can be determined by the expected entropy values of the pixels. As the progressive statistical model outputs the parameters of the value distribution functions, the expected entropy values can be calculated. However, this calculation is quite expensive. We chose to use an upper bound to approximate the expected entropy values. Let random variable X follow a mixture of logistic distributions determined by the number of mixtures K, mixture weights π_k, means μ_k and scales s_k where $k = 1, 2, \cdots, K$. An upper bound of the expected entropy of X can be defined by

$$U(X) = \sum_{k=1}^{K} \pi_k \log \pi_k + \sum_{k=1}^{K} \pi_k \left(\ln s_k + 2 \right). \tag{11}$$

The derivation of this upper bound can be found in the appendix. Given the approximation of the expected entropy values defined by Eq. 11, the pixels can be processed in an ascending order or descending order. Our experiments have shown that the descending order yields a better compression rate. In practice, this means that the system should process the most difficult pixels first. This counter-intuitive behavior actually helps to improve the overall compression rate since determining the most difficult pixels can make other pixels more predictable. This can be easily understood in the case of determining the borderline of two homogeneous areas.

4 Experiments

4.1 Impact of grouping method

We first evaluate the impact of the grouping methods described in Section 3.3. In this experiment, we trained the proposed probability model with different grouping methods using the training split of the CIFAR-10 dataset [21]. 3-scale architecture is used. For the dynamic grouping method, we partitioned the pixels into 3 groups at each scale according to the estimated entropy upper bound calculated using Eq. 11. We used the Adam optimization method with a learning rate of 1E-4 and a batch size of 128. Since CIFAR-10 is a dataset of small images (size 32x32), we trained the system until the training converged. Table 1 shows bits-per-pixel (BPP) results on the validation split of the CIFAR-10 dataset.

Table 1. Compression rate in bits-per-pixel on the validation split of the CIFAR-10 dataset using different grouping methods described in Section 3.3.

Random Grouping	Fixed Pattern (a)	Fixed Pattern (b)	Dynamic Grouping
10.83	10.66	**10.45**	10.60

Table 1 shows that an ordered grouping method performs better than the random grouping method. It also shows that the dynamic grouping method described in Section 3.3 has a better compression rate comparing to the fixed pattern (a) in Figure 3. It should be noted that the two methods have the same number of groups at each scale. According to the results, increasing the number of groups can significantly improve the compression rate as the fixed pattern (b), which contains 6 groups of pixels at each scale, outperforms all other grouping methods (3 groups) with a significant margin. In our experiments, we also noticed that the system with a fixed pattern grouping method converged faster than the dynamic grouping method.

4.2 Impact of the number of scales

In this experiment, we evaluated the impact of the number of scales using the CIFAR-10 dataset. We use the fixed pattern (a) as the grouping method. All other system parameters are the same except for the number of scales.

Table 2. Compression rate in bits-per-pixel on the validation split of the CIFAR-10 dataset using different scales.

2 scales	3 scales	4 scales	5 scales
10.95	10.66	10.614	**10.612**

The results in Table 2 show that increasing the number of scales can improve the compression rate. However, the improvement from 4 scales to 5 scales is negligible. Note that the tensor size at scale 4 accounts for less than 0.4% of the input image. Compressing this small amount of data does not bring much benefit to the overall compression rate.

4.3 Compression performance on the ImageNet64 and the OpenImage dataset

Next, we compare the proposed method against other state-of-the-art methods on the two main datasets that have been used to benchmark lossless image compression methods. The ImageNet64 dataset [22] contains images from the ImageNet dataset [23] that are downsampled to 64x64 pixels. The training split contains 1.28M images and the validation split contains 50K images. The OpenImage dataset [24] is a dataset with 9M images with annotations for various computer vision tasks. For a fair comparison, we prepared the training and the validation split of the OpenImage dataset in the same way as [13–15]. PNG format of the OpenImage dataset was used to avoid JPEG compression artifacts as suggested in [14].

We define three profiles, as shown in Table 3, for our lossless image compression system and evaluated their performance on the ImageNet64 and the OpenImage dataset. The difference between the "big" and the "normal" profile is the grouping method. Note that fixed pattern (b) has 6 groups at each scale while fixed pattern (a) has 3 groups. The "extra" profile defines a bigger system by increasing the number of scales from 3 to 4, the network size (the number of channels used in convolution layers) from 64 to 128, and the number of mixtures of the mixture of logistic distributions from 5 to 10. Note that the "big" profile does not increase the network size compared to the "normal" profile.

Table 3. The three profiles used to evaluate system performance.

	normal	big	extra
scales	3	3	4
grouping method	fixed pattern (a)	fixed pattern (b)	fixed pattern (b)
network size	64	64	128
number of mixtures	5	5	10

The system is trained with Adam optimization and learning rate 2×10^{-4} without weight decay. For the ImageNet64 dataset, the batch size is set to 64. For the OpenImage dataset, we use randomly cropped 128x128 patches from input images as the input. The batch size for the "normal" profile is 64, the "big" profile is 32, and the "extra" profile is 16. The learning rate is dropped by 10 times if a plateau is reached. The ImageNet64 dataset is trained for 40 epochs and the OpenImage dataset is trained for 20 epochs. All experiments had been executed on an NVIDIA DGX-1 system with Tesla V100 GPUs. All training was performed on a single GPU training mode.

The validation split of the OpenImage dataset contains 500 images randomly selected from the original validation set of the OpenImage dataset. The selected images are the same as those used in [13–15]. If an input image is too large to be processed because of the limitation of the GPU memory, we partition the image into patches and the bitstreams of all patches are concatenated to generate the final output file. The size of the patches is 496x496 for the "normal" and the "big" profile, and 256 for the "extra" profile. We note that this tiling operation degrades the compression rate because of the border effects. The reported compression rates of our methods are calculated from the size of the final output file which includes also extra metadata information such as image size and the number of scales.

Table 4 shows the compression results on the validation split of the ImageNet64 and the OpenImage dataset. To be compared with other literature, both bits-per-pixel and bits-per-subpixel (shown in parenthesis) are reported. The parameter column shows the number of parameters for the neural network-based methods.

Table 4. Compression results on the validation split of the ImageNet64 dataset and the OpenImage dataset. The numbers under the datasets columns are bits-per-pixel and bits-per-subpixel (in parenthesis). An empty cell indicates that the field is not relevant and "-" symbol indicates that the value is not reported by the authors.

	method	parameters	ImageNet64	OpenImage
Traditional	PNG [14]		17.22 (5.74)	12.09 (4.03)
	WebP [14]		13.92 (4.64)	9.09 (3.03)
	FLIF [14]		13.62 (4.54)	8.61 (2.87)
Flow-based	IDF [12]	84.33M	11.70 (3.90)	8.28 (2.76)
VAE-based	HiLLoC [10]	-	11.70 (3.90)	-
Context-based	RC [13]	-	-	8.37 (2.79)
	L3C [15]	5.01M	13.26 (4.42)	8.97 (2.99)
	SReC [14]	4.20M	12.90 (4.29)	8.10 (2.70)
	Ours (normal)	**1.87M**	11.89 (3.96)	8.14 (2.71)
	Ours (big)	**1.87M**	11.78 (3.93)	7.88 (2.63)
	Ours (extra)	9.93M	**11.33 (3.78)**	**7.48 (2.49)**

The results in Table 4 show that our methods significantly improve the compression rate on the ImageNet64 dataset with a much smaller model. The previous state-of-the-art method achieved 11.70 BPP with a model of 84.33M parameters and the proposed model achieved 11.33 BPP with a model of 9.93M parameters. On the OpenImage validation dataset, the "normal" profile achieves the same level of performance as the previous state-of-the-art methods with a much smaller model. The "big" and "extra" profile of the proposed method improves the state-of-the-art results by a big margin.

4.4 Encoding/decoding time

In this experiment, we evaluate the encoding and decoding time of the proposed method versus other lossless image compression methods. Methods that are based on pixel-wise autoregressive models are not included in the investigation since it is well-known that these methods are very slow and only capable of processing small images [14, 15, 25]. For L3C, SReC, and our methods, we selected an input image of size 256x256 and compressed/decompressed the image using different methods on the same system. Model loading time is not included in the measurement. For the IDF method, we recorded the encoding time using a pretrained CIFAR10 model on the CIFAR10 dataset. The LBB [9] method is a flow-based method that requires auxiliary bits to be sent together with the images to be encoded/decoded. It is included in the comparison since it works similar to pixel-wise statistical models that process pixels sequentially during the inference time. The values of the HiLLoC and the LBB methods are calculated from the numbers reported in [10] and [9] respectively. and We normalize the encoding/decoding time to be seconds per 32x32 pixels since the reported numbers in literature are for different sizes of images. Table 5 shows the encoding and decoding time in seconds. Note that our implementation is not optimized to achieve a fast encoding/decoding speed.

Table 5. Encoding/decoding time in seconds per 32x32 pixels.

	L3C	SReC	Ours	Ours	Ours	HiLLoC	IDF	LBB
	[15]	[14]	(normal)	(big)	(extra)	[10]	[12]	[9]
encoding	0.0078	0.025	0.031	0.052	0.074	0.159	0.430	64.4
decoding	0.0070	0.025	0.029	0.049	0.070	-	-	65.9

As shown in Table 5, the proposed method has the same level of speed as SReC [14] with the "normal" profile. The "big" and the "extra" profile show significant compression rate gains without heavily sacrificing the inference speed. The results also show that the proposed methods are also significantly faster than HiLLoC and IDF which also process pixels in groups. Compared to pixel-wise methods like LBB, our proposed methods have a tremendous advantage.

5 Discussions

In this paper, we presented a multi-scale progressive statistical model and a lossless image compression system based on it. The proposed statistical model efficiently balances the accuracy that pixel-wise models achieve and the speed that multi-scale models obtain. Experiments show that the proposed system outperforms the state-of-the-art methods by a significant margin on the two large benchmark datasets. The proposed system achieves superior performance with smaller models compared to other systems. The proposed system has a slightly higher computation complexity compared to other "fast" methods with significant gains in the compression rate. The proposed system incorporates a flexible mechanism where the pixel grouping method can be specified easily. We evaluated the static grouping methods using fixed patterns and a dynamic grouping method based on the upper bound of the estimated entropy. Experiments show that increasing the number of groups at each scale significantly improves the compression rate. The dynamic method shows certain gains compared to the static methods using fixed patterns when the number of groups at each scale is the same. However, the convergence speed at the training stage is much slower. This behavior needs to be further studied and a mechanism that can speed up the training with dynamic grouping methods shall be developed.

References

1. http://libpng.org/pub/png/libpng.html: (libpng Home Page)
2. Sneyers, J., Wuille, P.: FLIF: Free lossless image format based on MANIAC compression. In: 2016 IEEE International Conference on Image Processing (ICIP), IEEE (2016) 66–70
3. https://jpeg.org/jpeg2000/: (JPEG - JPEG 2000)
4. https://developers.google.com/speed/webp: (A new image format for the Web | WebP)
5. Sullivan, G.J., Ohm, J.R., Han, W.J., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. IEEE Transactions on circuits and systems for video technology **22** (2012) 1649–1668 Publisher: IEEE.
6. Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technical Journal **27** (1948) 379–423 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1948.tb01338.x.
7. Marpe, D., Wiegand, T., Sullivan, G.: The H.264/MPEG4 advanced video coding standard and its applications. IEEE Communications Magazine **44** (2006) 134–143 Conference Name: IEEE Communications Magazine.
8. Marpe, D., Schwarz, H., Wiegand, T.: Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Transactions on Circuits and Systems for Video Technology **13** (2003) 620–636 Conference Name: IEEE Transactions on Circuits and Systems for Video Technology.
9. Ho, J., Lohn, E., Abbeel, P.: Compression with Flows via Local Bits-Back Coding. arXiv:1905.08500 [cs, math, stat] (2020) arXiv: 1905.08500.
10. Townsend, J., Bird, T., Kunze, J., Barber, D.: HiLLoC: lossless image compression with hierarchical latent variable models. (2019)

11. Johnston, N., Vincent, D., Minnen, D., Covell, M., Singh, S., Chinen, T., Hwang, S.J., Shor, J., Toderici, G.: Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks. arXiv:1703.10114 [cs] (2017) arXiv: 1703.10114.
12. Hoogeboom, E., Peters, J.W.T., Berg, R.v.d., Welling, M.: Integer Discrete Flows and Lossless Compression. arXiv:1905.07376 [cs, stat] (2019) arXiv: 1905.07376.
13. Mentzer, F., Van Gool, L., Tschannen, M.: Learning Better Lossless Compression Using Lossy Compression. arXiv:2003.10184 [cs, eess] (2020) arXiv: 2003.10184.
14. Cao, S., Wu, C.Y., Krähenbühl, P.: Lossless Image Compression through Super-Resolution. arXiv:2004.02872 [cs, eess] (2020) arXiv: 2004.02872.
15. Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., Van Gool, L.: Practical Full Resolution Learned Lossless Image Compression. arXiv:1811.12817 [cs, eess] (2019) arXiv: 1811.12817.
16. Oord, A.v.d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., Kavukcuoglu, K.: Conditional Image Generation with PixelCNN Decoders. arXiv:1606.05328 [cs] (2016) arXiv: 1606.05328.
17. Oord, A.v.d., Kalchbrenner, N., Kavukcuoglu, K.: Pixel Recurrent Neural Networks. arXiv:1601.06759 [cs] (2016) arXiv: 1601.06759.
18. Salimans, T., Karpathy, A., Chen, X., Kingma, D.P.: PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. (2016)
19. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using Real NVP. arXiv:1605.08803 [cs, stat] (2017) arXiv: 1605.08803.
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs] (2015) arXiv: 1512.03385.
21. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. (2009) Publisher: Citeseer.
22. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. arXiv:1707.08819 [cs] (2017) arXiv: 1707.08819.
23. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09. (2009)
24. Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., Ferrari, V.: The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. IJCV (2020)
25. Ramachandran, P., Paine, T.L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., Hasegawa-Johnson, M.A., Campbell, R.H., Huang, T.S.: Fast Generation for Convolutional Autoregressive Models. arXiv:1704.06001 [cs, stat] (2017) arXiv: 1704.06001.