Supplementary Material for "LC-FDNet: Learned Lossless Image Compression with Frequency Decomposition Network"

Hochang Rhee[1], Yeong Il Jang[1], Seyun Kim[2], Nam Ik Cho[1]
[1]Department of ECE, INMC, Seoul National University, Seoul, Korea
[2]Gauss Labs Inc.
hochang,jyicu@ispl.snu.ac.kr, seyun.kim@gausslabs.ai, nicho@snu.ac.kr

1. Experiments on Low-Resolution Dataset

Here, we compare our method with other compression algorithms on low-resolution dataset in Table. 6. We validate on three benchmark dataset, ImageNet32 [3], ImageNet64 [3] and CIFAR10 [6]. We do not retrain our network on the low-resolution dataset and rather use the network trained on Flickr2K.

Compared to the non-learning methods, our method shows superior performance for all three datasets, even though our network is trained on high-resolution dataset. For the learning-based methods, it can be observed that PixelCNN [8] achieves the best performance on all three datasets, followed by MS-PixelCNN [10]. However, as can be seen in Table 7, these methods require impractical inference time prohibiting them from practical use. In addition, although Zhang *et al.* [13] and IDF [4] show competitive performance to ours, they require at least 1.7 times more computation time.

Table 6. Comparison of our method with other non-learning and learning-based codecs on low-resolution benchmark dataset. We measure the performances in bits per pixel (bpp). Best performance is highlighted in bold.

Method	ImageNet32	ImageNet64	CIFAR10
PNG [2]	19.17	17.22	17.67
JPEG2000 [9]	19.05	13.52	15.60
WebP [12]	15.84	13.92	15.42
LCIC [5]	15.27	14.18	15.90
FLIF [11]	13.56	13.62	12.57
JPEG-XL [1]	19.17	16.18	17.67
PixelCNN [8]	**11.49**	**10.71**	**9.42**
MS-PixelCNN [10]	11.85	11.10	-
Zhang *et al.* [13]	-	11.89	10.45
IDF [4]	12.54	11.70	10.02
L3C [7]	14.28	13.26	-
Ours	13.19	12.85	11.32

Table 7. Encoding time in seconds required for 512×512 images.

PixelCNN	MS-PixelCNN	Zhang *et al.*	IDF	L3C	Ours
30600	300	1.47	20.40	0.37	0.84

2. Network Architectures

In this Section, we present the architecture detail of LFC and HFC in Fig. 6.

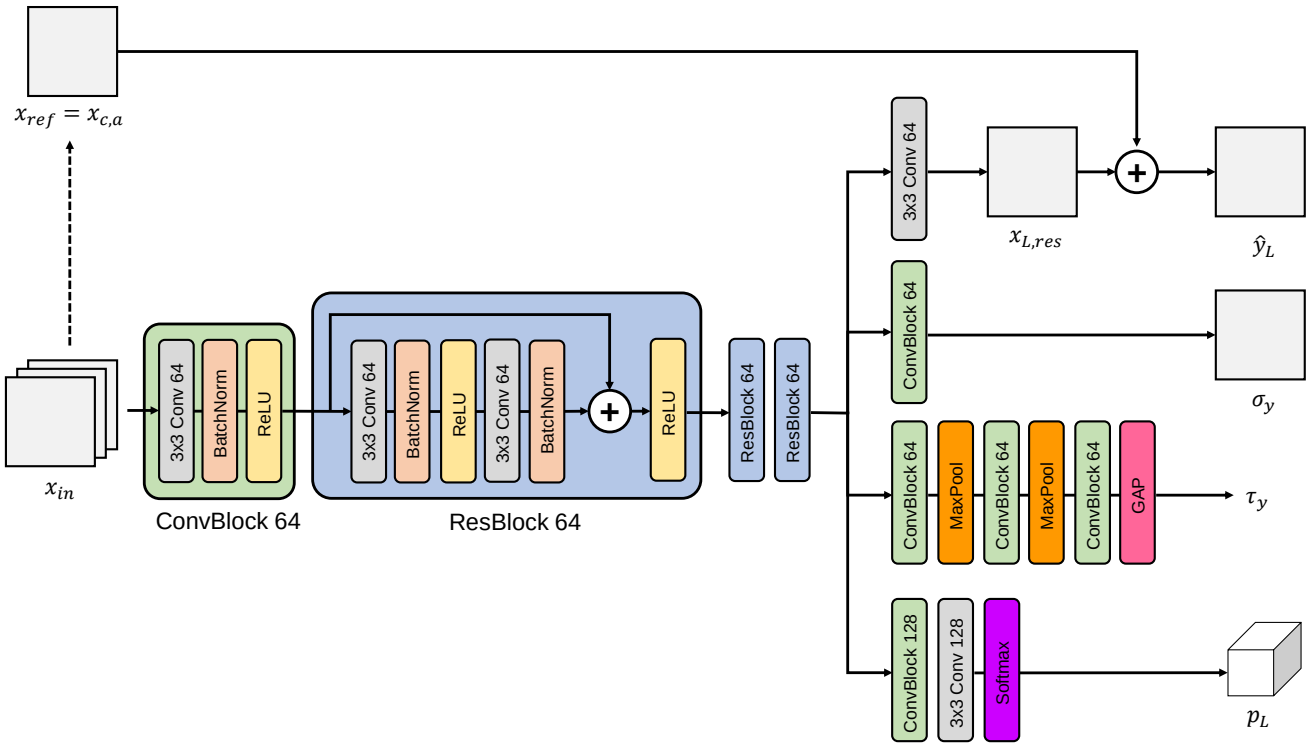
LFC: Given input x_{in}, the LFC generates four outputs: 1) subimage prediction $\hat{y}_L$, 2) error variance map σ_y, 3) error variance threshold τ_y, and 4) probability distribution p_L. Note that the dimension of x_{in} differs depending on the subimage. We initially pass the input through one ConvBlock and three ResBlocks and generate an intermediate feature.

For the prediction of a subimage, we pass the intermediate feature through one convolutional layer. We adopt a residual scheme for the subimage prediction. That is, instead of directly estimating the pixel value of the subimage y, the network estimates $x_{L,res}$, which is the difference between y and a reference image x_{ref} (set as $x_{c,a}$). For instance, in the case of compressing $y = x_{Y,d}$, the network sets $x_{Y,a}$ as offset and estimates the residual between $x_{Y,a}$ and $x_{Y,d}$. The final output $\hat{y}_L$ is derived as $x_{L,res} + x_{Y,a}$. In other words, the network estimates the residual in respect to x_{ref}. This leads to stable training and performance enhancement.

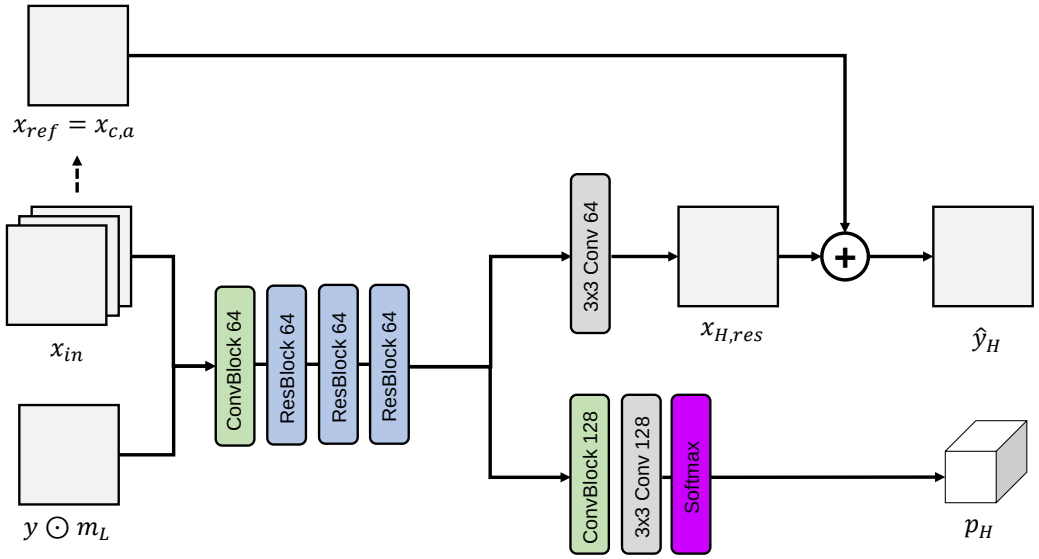
To obtain the error variance map σ_y, we pass the intermediate feature through one ConvBlock. For the error variance threshold τ_y, the intermediate feature goes through a sequence of ConvBlock and MaxPooling. At the end, global average pooling (GAP) is applied to derive τ_y, which is a scalar value.

HFC: The architecture of the HFC is similar to the LFC. The difference is that the low-frequency components are given as additional input, and the HFC generates only two outputs: 1) subimage prediction $\hat{y}_H$ and 2) probability distribution p_H. Other factors are the same as the LFC.

Finally, the probability distribution p_L is derived by



(a) Low Frequency Compressor Network Architecture



(b) High Frequency Compressor Network Architecture

Figure 6. Architecture details of LFC and HFC.

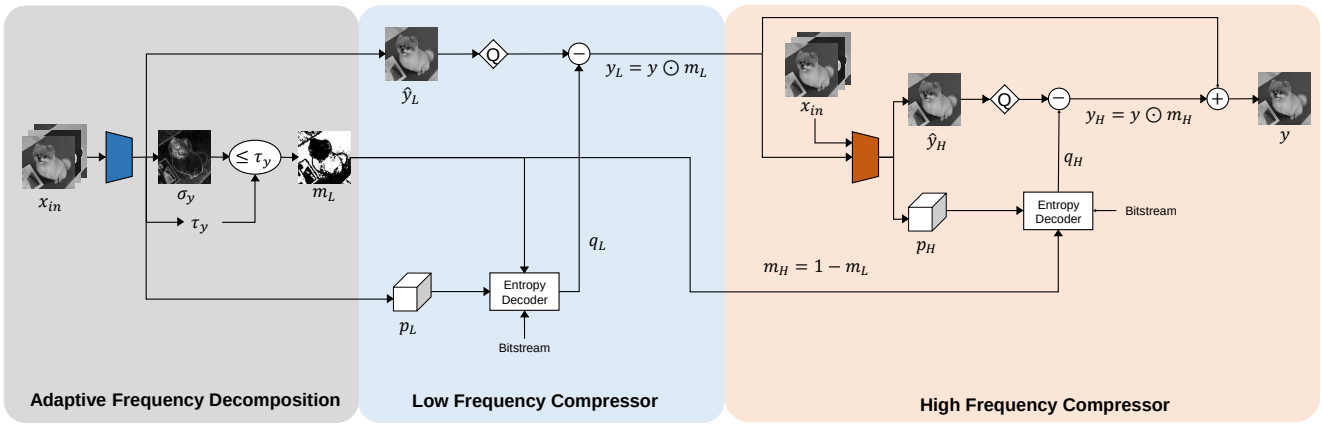


Figure 7. Illustration of the decoding procedure.

passing the intermediate feature through one ConvBlock, one convolutional layer, and a softmax operation at the end. Note that whereas other layers consist of 64 hidden units, we double the number of hidden units for obtaining p_L.

3. Decoding Procedure

In Fig. 7, we explain the procedure of decoding a bitstream into a subimage. When encoding a subimage y, the network compresses the subimage into bitstreams with the inputs x_{in} and y. For the decoding, the inputs are x_{in} and bitstreams. We first explain the decoding of the low-frequency components in AFD and LFC parts.

Same as in the encoding procedure, four outputs are generated from x_{in}: 1) subimage prediction $\hat{y}_L$, 2) error variance map σ_y, 3) error variance threshold τ_y, and 4) probability distribution p_L. ADF is also equivalent as in the encoding procedure, where the low-frequency mask m_L is obtained using Eq. 2. Afterward, the entropy decoder receives 3 inputs p_L, m_L, and bitstream, and then generates $q_L \in \mathcal{R}^{\frac{H}{2} \times \frac{W}{2} \times 1}$. From the obtained q_L, we reconstruct y_L, the low-frequency component of the subimage y. Specifically, $y_L = round(\hat{y}_L) - q_L$, which is the inverse of the encoding procedure $q_L = round(\hat{y}_L - y_L)$.

Decoding of high-frequency components in the HFC is similar to the LFC. Precisely, x_{in} and $y_L = y \odot m_L$ are given as input to produce $\hat{y}_H$ and p_H. Again, the entropy decoder receives three inputs p_H, m_H, and bitstream, and generates q_H. Then, the high-frequency components of the subimage y_H is reconstructed from $y_H = round(\hat{y}_H - q_H)$. Finally, we reconstruct the subimage y by the addition of y_L and y_H.

4. Subimage Order Analysis

In this section, we show how the performance differs depending on the order of subimage compression. Specifically, we compare the order of $a \rightarrow d \rightarrow b \rightarrow c$ (ours) against $a \rightarrow b \rightarrow c \rightarrow d$ (MS-PixelCNN [10]) in Table 8. We observe that our design of subimage order achieves at most 1.8% performance gain. In Table 9, we report the compression result for each subimage when compressed in the order of $a \rightarrow b \rightarrow c \rightarrow d$. Compared to Table 2, it can be observed that whereas d in $a \rightarrow d \rightarrow b \rightarrow c$ requires 2.13 bpp, b in $a \rightarrow b \rightarrow c \rightarrow d$ requires 2.23 bpp. From this, we can conclude that given a, the estimation of d is easier than b.

Table 8. Comparison of compression performance with different order of subimage compression.

Method	CLIC.m	CLIC.p	DIV2K
$a \rightarrow b \rightarrow c \rightarrow d$	4.72 +1.3%	5.36 +1.5%	5.59 +1.8%
$a \rightarrow d \rightarrow b \rightarrow c$	4.66	5.28	5.49

Table 9. Compression result of each subimage for the DIV2K dataset when compressed in the order of $a \rightarrow b \rightarrow c \rightarrow d$.

bpp	a	b	c	d
Y	-	1.02	0.81	0.77
U	-	0.61	0.45	0.45
V	-	0.60	0.45	0.43
Total	2.68	2.23	1.71	1.65

References

[1] Jyrki Alakuijala, Ruud van Asseldonk, Sami Boukortt, Martin Bruse, Iulia-Maria Comșa, Moritz Firsching, Thomas Fischbacher, Evgenii Kliuchnikov, Sebastian Gomez, Robert Obryk, et al. Jpeg xl next-generation image compression architecture and coding tools. In *Applications of Digital Image Processing XLII*, volume 11137, page 111370K. International Society for Optics and Photonics, 2019. 1

[2] Thomas Boutell and T Lane. Png (portable network graph-

ics) specification version 1.0. *Network Working Group*, pages 1–102, 1997. 1

[3] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017. 1

[4] Emiel Hoogeboom, Jorn WT Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. *arXiv preprint arXiv:1905.07376*, 2019. 1

[5] Seyun Kim and Nam Ik Cho. Hierarchical prediction and context adaptive coding for lossless color image compression. *IEEE Transactions on image processing*, 23(1):445–449, 2013. 1

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1

[7] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10629–10638, 2019. 1

[8] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016. 1

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

[10] Scott Reed, Aäron Oord, Nal Kalchbrenner, Sergio Gómez Colmenarejo, Ziyu Wang, Yutian Chen, Dan Belov, and Nando Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning*, pages 2912–2921. PMLR, 2017. 1, 3

[11] Jon Sneyers and Pieter Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE international conference on image processing (ICIP)*, pages 66–70. IEEE, 2016. 1

[12] WebEngines Blazer Platform Version. 1.0 hardware reference guide, xp-002202892, network engines. *Inc., Jun*, 1:92, 2000. 1

[13] Honglei Zhang, Francesco Cricri, Hamed R Tavakoli, Nannan Zou, Emre Aksu, and Miska M Hannuksela. Lossless image compression using a multi-scale progressive statistical model. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 1