BlockCNN: A Deep Network for Artifact Removal and Image Compression

Danial Maleki, Soheila Nadalian, Mohammad Mahdi Derakhshani, Mohammad Amin Sadeghi
School of Electrical and Computer Engineering, University of Tehran
{d.maleki, nadalian.soheila, mderakhshani, asadeghi}@ut.ac.ir

Abstract

We present a general technique that performs both artifact removal and image compression. For artifact removal, we input a JPEG image and try to remove its compression artifacts. For compression, we input an image and process its 8×8 blocks in a sequence. For each block, we first try to predict its intensities based on previous blocks; then, we store a residual with respect to the input image. Our technique reuses JPEG's legacy compression and decompression routines. Both our artifact removal and our image compression techniques use the same deep network, but with different training weights. Our technique is simple and fast and it significantly improves the performance of artifact removal and image compression.

1. Introduction

The advent of Deep Learning has led to multiple breakthroughs in image representation including: super-resolution, image compression, image enhancement and image generation. We present a unified model that can perform two tasks: 1- artifact removal for JPEG images and 2- image compression for new images.

Our model uses deep learning and legacy JPEG compression routines. JPEG divides images into 8×8 blocks and compresses each block independently. This causes block-wise compression artifact (Figure 2). We show that the statistics of a pixel's artifact depends on where it is placed in the block (Figure 2). As a result, an artifact removal technique that has a prior about the pixel's location has an advantage. Our model acts on 8×8 blocks in order to gain from this prior. Also, this let us reuse JPEG compression.

For image compression, we examine image blocks in a sequence. When each block is being compressed, we first try to predict the block's image according to its neighbouring blocks (Figure 1). Our prediction has a residual with respect to the original block. We store this residual which requires less space than the original block. We compress this residual using legacy JPEG techniques. We can trade off quality versus space using JPEG compression ratio.

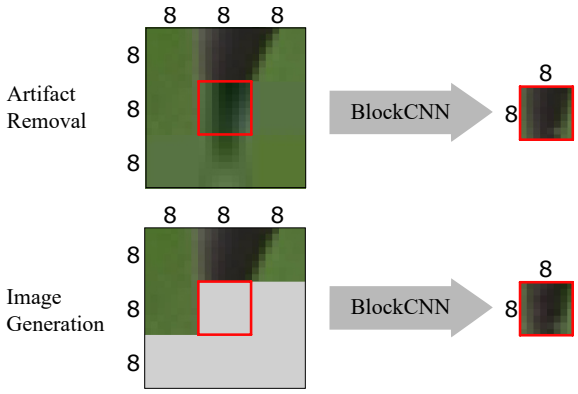


Figure 1. BlockCNN: This architecture can be used for both artifact removal and image compression. BlockCNN acts on 8×8 image blocks. Top: To remove artifact from each block, we input this block together with its eight adjacent blocks and try to removes artifacts from the center block. Bottom: This architecture can predict a block given four of its neighbors (three blocks to the top and one to the left). We use this image prediction to compress an image. We first try to predict a block and then we store the residual which takes less space.

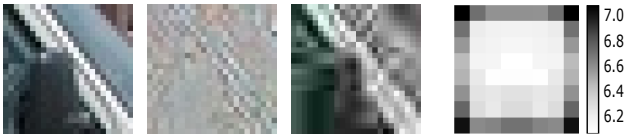


Figure 2. Left: JPEG compresses each 8×8 block independently. Therefore, each block has independent artifact characteristics. Our artifact removal technique acts on each block separately. Right: The statistics of a pixel's compression artifact depends on where it is located within an 8×8 block. The right figure, illustrates Mean Square Error of pixel intensities (within a block) after compression. We used 6 Million image blocks with quality factor of 20 to produce this figure.

Our image prediction is a deterministic process. Therefore, during decompression, we first try to predict a block's content and then add up the stored residual. After decompression, we perform artifact removal to further improve the quality of the restored image. With this technique we get a superior quality-space trade-off.

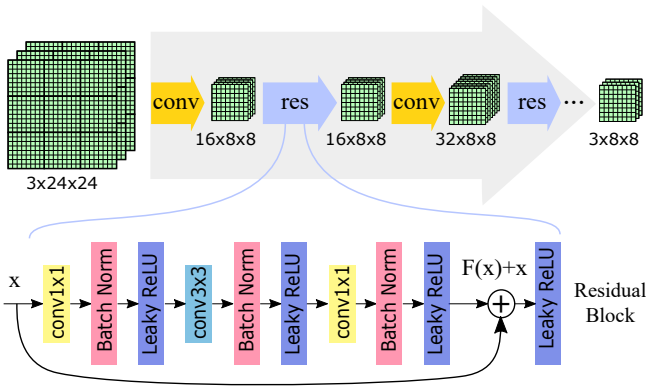


Figure 3. Our network architecture. Top: We input a 24×24 color image and output an 8×8 color image. Our network has a series of convolution and residual blocks. Bottom: Our residual block consists of several operations including convolution, batch normalization, and leaky ReLU activation function.

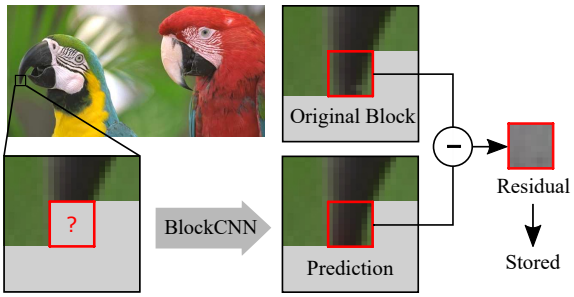


Figure 4. Our compression pipeline. We process image blocks in a sequence. For each block (highlighted with question mark), we first try to predict its intensities using the previous blocks. Then we compute the residual between our prediction and the original block. We store this residual and continue to the next block. During decompression we go through a similar sequential process. We first predict an image block using its previous blocks and then add up the residual.

2. Related Work

JPEG [19] compresses 8×8 blocks using quantized cosine coefficients. JPEG compression could lead to unwanted compression artifacts. Several techniques are developed to reduce artifacts and improve compression:

- **Deep Learning**: Jain et al. [10] and Zhang et al. [21] trained a network to reduce Gaussian noise. Dong et al. [5] trained a network to reduce JPEG compression artifacts. Mao et al. [12] developed an encoder-decoder network for denoising. Theis et al. [15] presented an auto-encoder based compression technique.
- **Residual-based Techniques**: Svoboda et al. [14] applied residual representation learning to define an easier task for network. Baig et al. [2] use image inpainting before compression. Dong et al. [5] reuse pretrained models to speeds up learning.
- **Generative Techniques**: Santurkar et al. [13] used Deep Generative models to reproduce image and video and remove artifacts. Galteri et al. [7] used Generative Adversarial Networks to reduce compression artifact. A notable work in image generation is PixelCNN by Oord et al. [18]. Dahl et al. [4] introduced a super-resolution technique based on PixelCNN. Our BlockCNN architecture is also inspired by PixelCNN.
- **Recurrent Neural Networks**: Toderici et al. [17] presented a compression technique using an RNN-based encoder and decoder, binarizer, and a neural network for entropy coding. They also employ a new variation of Gated Recurrent Unit [3]. Another work by Toderici et al. [16] proposed a variable-rate compression technique using convolutional and deconvolutional LSTM [9] network.

3. BlockCNN

Similar to JPEG, we partition an image into 8×8 blocks and process each block separately. We use a convolutional neural network that inputs a block together with its adjacent blocks (a 24×24 image), and outputs the processed block in the center. We call this architecture BlockCNN. We use BlockCNN both for artifact removal and image compression. In the following subsections, we discuss the specifications in more detail.

3.1. Deep Architecture

BlockCNN consists of a number of convolution layers in the beginning followed by a number of residual blocks (resnet [8]). A simplified schematic of the architecture is illustrated in Figure 3. A residual block is formulated as:

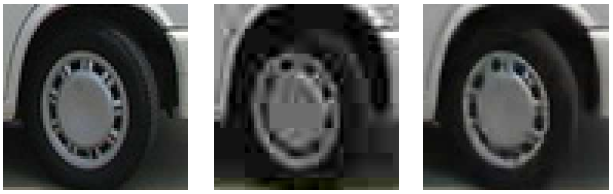
$$G(x) = F(x) + x \tag{1}$$

where x shows the identity mapping and $F(x)$ is a feed-forward neural network trying to learn the residual (Figure 3, bottom). Residual blocks avoid over-fitting, vanishing gradient, and exploding gradient. Our experiments show that residual blocks are superior in terms of accuracy and rate of convergence.

We use mean square error as loss function. For training, we use Adam [11] with weight decay of 10^{-4} and learning rate of 10^{-3}. We train our network for $120,000$ iterations.

3.2. BlockCNN for Artifact Removal

For artifact removal we train BlockCNN with JPEG compressed blocks as input and an uncompressed block as target (Figure 1). This network has three characteristics that make it successful. **Residual**: Since compression artifact is naturally a residual, predicting artifact as residual is easier than encoding and decoding a block. It leads to faster



Original Image Compressed Image Enhanced Image

Figure 5. Left: Original image before compression. Center: JPEG compressed Image with block-wise artifact. Right: Enhanced Image using our technique. Note that our result has improved blocking and ringing artifacts.

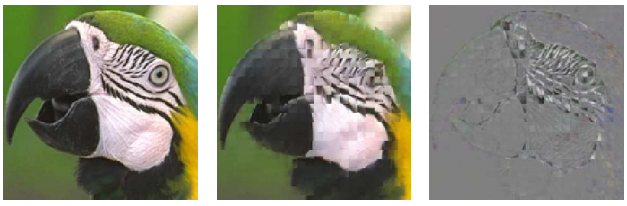


Figure 6. Left: Original Image fed to BlockCNN for compression. Center: BlockCNN prediction. Each block in this image shows the best prediction using previously seen blocks. Right: The difference between the original image and our prediction (Residual). We store residual instead of the original image.

and improved convergence behavior. **Context**: BlockCNN input adjacent cells so it can use context to improve its estimate of residual. Of course, larger context can improve performance but we limit context to one block for better illustration. **block structure**: The statistics of artifact depend on where a pixel is placed within a block. BlockCNN takes advantage of the prior of pixels' location within a block.

3.3. BlockCNN for Image Compression

The idea behind our technique is the following: Anything that can be predicted does not need to be stored.

We traverse image blocks row wise. Given each block, we first try to predict its intensities given previously seen blocks. Then we compute the residual between our prediction and the actual block and store this residual (Figure 4). When storing a block's residual we compress it using JPEG. JPEG has two major benefits: 1- It is simple, fast and readily available; 2- We can reuse JPEG's quality factor to trade off size with quality.

During decompression we follow the same deterministic process. For each block, we try to predict its intensities and then add up the stored residual. Note that for predicting a block's image, we use the *compressed* version of the previous blocks, because otherwise compression noise propagates and accumulates throughout image. For this prediction process we reuse BlockCNN architecture but with different training examples and thus different weights. The major difference in training examples is that only four out of nine blocks are given as input.

Ours JPEG AR-CNN

Figure 7. Qualitative comparison between artifact removal algorithms.

4. Experiments and Results

We used PASCAL VOC 2007 [6] for training. Since JPEG compresses images in Lab colorspace, it produces separate intensity and color artifacts. Therefore, we perform training in Lab colorspace to improve our performance for ringing, blocking, and chromatic distortions.

For optimization process we use Adam [11] with weight decay of 10^{-4} and a learning rate of 10^{-3}. This network is trained for $120,000$ iterations.

4.1. Artifact Removal Results

Artifact removal techniques in the literature are usually benchmarked using LIVE [1] dataset. Peak signal-to-noise ratio (P-SNR) and structural similarity index measurement (SSIM) [20] are regularly used as evaluation criteria. We follow this convention and compare our results with AR-CNN [5] (Figure 7). We trained a BlockCNN with 9 residual blocks. We use approximately 6 Million pairs of input and target for training.

4.2. Image Compression Results

Kodak [1] dataset is commonly used as a benchmark for Image Compression. We evaluate our method using peak signal-to-noise ratio (P-SNR) and structural similarity index measurement (SSIM). After image decompression, we perform artifact removal to improve quality. We compare our results with Toderici et al. [17] and CAE [15] (Figure 9).

5. Discussion

We presented BlockCNN, a deep architecture that can perform artifact removal and image compression. Our technique respects JPEG compression conventions and acts on 8×8 blocks. The idea behind our image compression technique is that before compressing each block, we try to predict as much as possible from previously seen blocks. Then, we only store a residual that takes less space. Our technique reuses JPEG compression routines to compress the residual. Our technique is simple but effective and it beats baselines for high compression ratios.

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

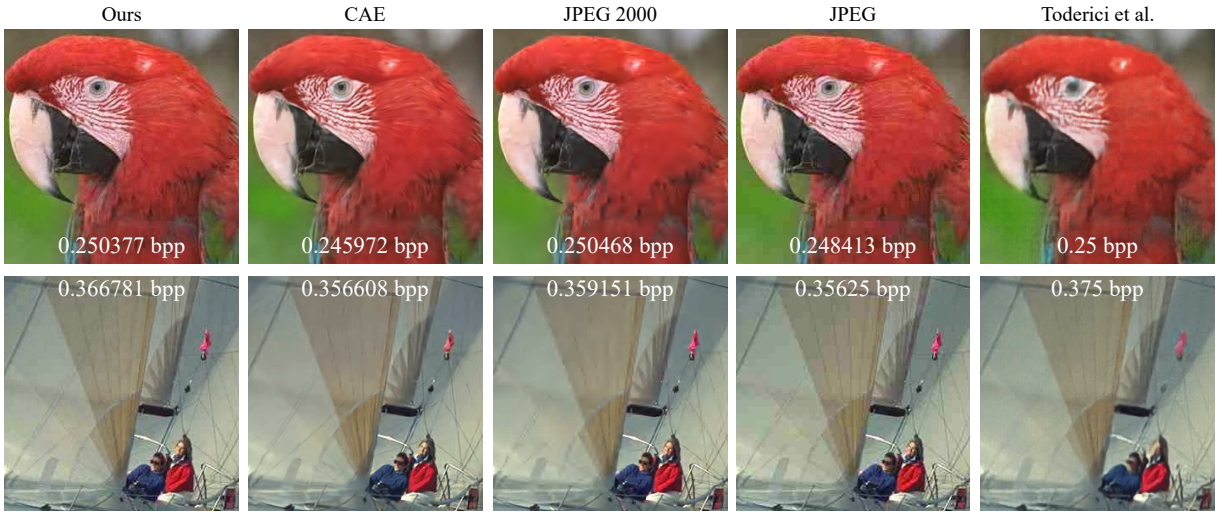


Figure 8. Qualitative comparison between different compression algorithms at low bit rates. Note that our technique performs a better job at preserving details.

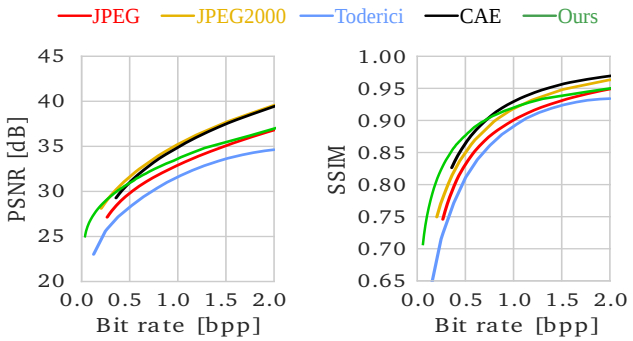


Figure 9. Comparison of compression technqiques using PSNR and SSIM. Note that our technique outperforms baselines at low bit rates.

References

[1] Live image quality assessment database.

[2] M. H. Baig, V. Koltun, and L. Torresani. Learning to inpaint for image compression. *CoRR*, abs/1709.08855, 2017.

[3] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[4] R. Dahl, M. Norouzi, and J. Shlens. Pixel recursive super resolution. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*.

[5] C. Dong, Y. Deng, C. C. Loy, and X. Tang. Compression artifacts reduction by a deep convolutional network. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2), June 2010.

[7] L. Galteri, L. Seidenari, M. Bertini, and A. D. Bimbo. Deep generative adversarial compression artifact removal. *CoRR*, abs/1704.02518, 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[10] V. Jain and S. Seung. Natural image denoising with convolutional networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*. Curran Associates, Inc., 2009.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] X. Mao, C. Shen, and Y. Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Advances in Neural Information Processing Systems 2016*.

[13] S. Santurkar, D. M. Budden, and N. Shavit. Generative compression. *CoRR*, abs/1703.01467, 2017.

[14] P. Svoboda, M. Hradis, D. Barina, and P. Zemcík. Compression artifacts removal using convolutional neural networks. *CoRR*, abs/1605.00366, 2016.

[15] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[16] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. *CoRR*, 2015.

[17] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. In *CVPR , 2017*.

[18] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[19] G. K. Wallace. The jpeg still picture compression standard. *Commun. ACM*, 34(4), Apr. 1991.

[20] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image-qualityassessment:fromerrorvisibilitytostructuralsimilarity. *Trans. Img. Proc.*, 13(4):600–612, Apr. 2004.

[21] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. *CoRR*, 2016.