Learning to Hide Residual for Boosting Image Compression

Yi-Lun Lee[1]
vongola850704@gmail.com
Yen-Chung Chen[1]
yenc.cs06g@nctu.edu.tw
Min-Yuan Tseng[1]
piews482zt@gmail.com
Yi-Hsuan Tsai[2]
wasidennis@gmail.com
Wei-Chen Chiu[1]
walon@cs.nctu.edu.tw

[1] National Yang Ming Chiao Tung University
[2] Phiar Technologies, Inc.

Abstract

Lossy compression usually leads to severe compression artifacts, such as blocking boundary, mosquito noise, and blur. Reducing compression artifacts is essential for better visual experience and quality when transmitting data under limited bandwidth, where the sender compresses an image and transmits it via a communication channel to the receiver side. To tackle this problem, most existing methods aim to directly recover details from received compressed image, instead of fully exploiting the rich information contained in the uncompressed image. In this paper, we focus on leveraging the residual information, i.e. the difference between a compressed image and its corresponding original/uncompressed one, and propose to hide the residual into the original image by a novel framework. As such, our model that considers this resultant image with the hidden information has a better ability to recover the residual caused by the compression process. Afterwards, the hidden residual could be decoded from the received image and used to boost the quality of image reconstruction on the receiver side. Extensive experiments verify the efficacy of our proposed framework in reducing compression artifacts and showing favorable performance against numerous baselines.

1 Introduction

Image has long been an indispensable part of our daily lives in which people record events, share knowledge, and the media of communication. Along with the development of recording technology and the demand for better visual experience, the resolution of image data grows higher and higher which naturally leads to larger size. However, since the storage space is not limitless and the communication channel is usually with constrained bandwidth, image compression is crucial for data transmission and data storage.

Decades of efforts have been devoted to advance the image compression techniques and bring out various image coding standards, such as BPG [3], JPEG-2000 [1], and JPEG [17]. Without loss of generality, most compression methods belong to the lossy compression scheme, where the file size is reduced by discarding redundant information or some image details. Normally, the degradation in image quality caused by compression should be unnoticeable by end-users. However, as the demand of transmitting and storing more data grows (e.g. real-time communication), having artifacts in the compressed images becomes an inevitable issue due to the heavy compression process.

Given a general scenario where a sender first compresses an image and transmits it to a receiver, the visual difference between a compressed image and the corresponding original/uncompressed one is known as *residual*. Various approaches [7, 14, 19] are proposed to compensate the visual difference by having post-processing models to estimate the possible residual from the compressed image; while we also witness the work [16] that aims to directly transmit the residual in a compact form along with the compressed image. This work has the advantage of preserving the real residual, while the post-processing approaches could produce the estimated residual that could be quite different from the original one. However, [16] requires non-negligible bitrates to transmit the residual. Moreover, it would need additional efforts to design the compression approach (e.g. quantization and entropy encoding) and modify the communication protocol to support the residual transmission.

We advance along the direction of dealing with the real residual in this paper. We propose a novel framework which needs not transmit the residual separately from the data stream of the compressed image, and instead we directly hide the residual into the original image, prior to the image compression. As such, we are no longer concerned about the transmission of the residual. Moreover, our framework learns to hide useful residual information into the original image in a way that we can still successfully decode back the residual related to the hidden information after applying compression to the resultant image.

Particularly, our framework is orchestrated to boost the visual quality of images compressed by any off-the-shelf compression methods, in which it can be treated as a compression enhancement approach. In order to show the flexibility and generalizability of our proposed method, we experiment upon different image compression methods, including JPEG, JPEG-2000, BPG, CAE-B [20], CAE-P [20], and RNN [15], where the last three are deep-learning-based compression models. We conduct extensive experiments on Kodak [11] and Kinetics [9] datasets under various settings, and the quantitative evaluation demonstrates the efficacy of our method in comparison to several post-processing-based approaches.

2 Related Work

As motivated above, in comparison to the works of developing a whole new compression codec from scratch (e.g. [2], [13], [20], [15]) or the ones transmitting the residual information explicitly (e.g. [16]), our method proposed can be integrated with any off-the-shelf compression codecs and needs no additional transmission of residual, in which such scheme is actually similar to the post-processing methods.

Basically, the post-processing methods aim to reduce compression artifacts at the receiver/decoder end directly. Earlier approaches such as [5] adopt adaptive filters in order to simultaneously smooth out the artifacts and preserve the image details, while [4] propose a two-step method that learns a dictionary to represent typical features of compressed JPEG image and favors the decompressed procedure. To further enhance the performance, deep-

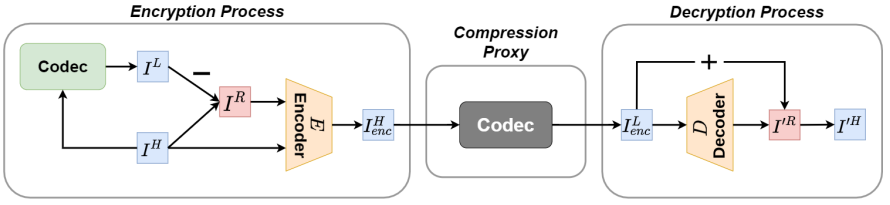

Figure 1: Overview of the proposed framework. Starting from the encryption process, the encryption encoder E hides the residual I^R into the original image I^H and outputs the encoded image I^H_{enc}, where I^R is the difference between I^H and its compression I^L produce by a specific compression codec (e.g. JPEG or BPG). I^H_{enc} is then fed into the compression codec component and obtain its compressed version I^L_{enc}. Followed by the decryption process, the original image is reconstructed as I'^H by combining I^L_{enc} with the decoded hidden information I'^R via the decryption decoder D.

learning-based methods are developed recently. [12] utilize skip connection, which helps not only recover clean image details but also the training process of deep convolutional neural networks. Built upon SR-CNN [6], [7] focus on feature enhancement, which maps noisy low-level features to another learnt feature space for eliminating compression artifacts. With the concept of residual learning, [19] propose a deep CNN model (DnCNN) by adding batch normalization in order to handle multiple image restoration tasks, e.g., JPEG de-blocking. On the other hand, [14] propose a deep persistent network (MemNet) which utilizes proposed memory blocks to mine persistent memory via an adaptive learning process.

Our proposed method in this paper shares a similar goal to the above-mentioned approaches that aim to post-process the compressed image and reconstruct back an output with less artifacts, but clearly distinguish itself from the others by having a new solution of hiding the *residual* information prior to the image compression process, which provides effective cues for our decoder to reduce the output artifacts. In addition, since we take advantages of existing image compression codecs (i.e. explicitly considering the compression codecs into our framework training), our proposed method can be applicable to various types of codecs and boost them, including the traditional and deep-learning-based ones.

3 Proposed Method

Our goal is learning to hide the residual in the original/uncompressed image where the hidden information should still exist after compression, such that the residual could be decoded back on the receiver side to further boost the visual quality of the reconstructed image. To this end, we propose a framework consisting of three components, as depicted in Fig. 1: 1) an encryption process, 2) compression proxy, and 3) a decryption process. Note that the detailed architecture of the networks used in these three components are provided in the supplementary material. We now provide more details for each component in our proposed framework in the following subsections.

3.1 Encryption Process

Given an image compression method such as BPG to compress an image I^H, our encryption encoder E aims to hide the residual information I^R, i.e. the difference between an image I^H and its compressed version I^L, into the image I^H. The encryption encoder E takes the residual

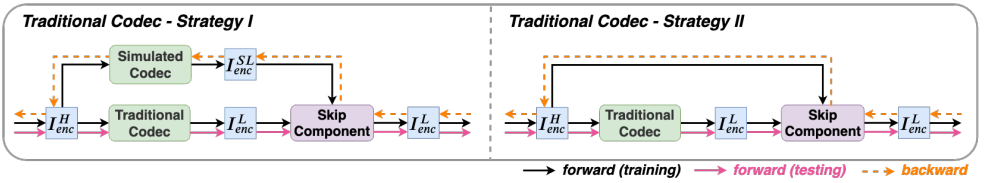

Figure 2: When the compression codec belongs to the deep-learning-based codec, we can directly adopt it as the compression proxy component during model training. Instead, if the traditional codec is used during compression, two different strategioes (i.e. Strategy I and Strategy II) are proposed as the compression proxy, in order to enable the end-to-end training. For both strategies, we provide the forward and backward flows during training, as well as the the forward flow for testing. Please refer to Section 3.3 for more details.

I^R and original/uncompressed image I^H as the input, and outputs an encoded uncompressed image I^H_{enc} which contains hidden residual information, i.e. $I^H_{enc} = E\left(I^H, I^R\right)$. Please refer to the left of Fig. 1 for the illustration of our encryption process.

Now the I^H_{enc} becomes the target that we would like to compress on the sender side and transmit afterwards to the receiver side. Assume the communication channel is lossless, i.e. the data is never lost during the transmission, the receiver side will obtain a compressed version of I^H_{enc}, which is denoted as I^L_{enc}.

3.2 Decryption Process

When I^L_{enc} arrives the receiver side, our decryption decoder D attempts to decrypt from I^L_{enc} the hidden information, which is denoted as I'^R. As I'^R is related to the residual, i.e. the difference between I^L and I^H caused by the compression, we hence expect to well recover the original input image I^H by adding up I^L_{enc} and I'^R. Here, we denote the reconstruction produced by $I^L_{enc} + I'^R$ as I'^H. Although I'^R is not exactly the same as I^R (as I^L_{enc} is not identical to I^L), the entire network is end-to-end trainable, so that the encoder can still learn to encrypt useful information of I^R into I^H_{enc}. Please refer to the right of Fig. 1 for the illustration of our decryption process.

3.3 Compression Proxy

As illustrated in Fig. 1, the connection between I^H_{enc} and I^L_{enc} is the compression codec (e.g. BPG or JPEG), which compresses I^H_{enc} into I^L_{enc}. We denote the compression between I^H_{enc} and I^L_{enc} as the *compression proxy*. In general, compression codecs could be categorized into two groups: deep-learning-based codec and traditional codec. Since the deep-learning-based codec is differentiable, it can be simply integrated with the encryption and decryption processes to construct an end-to-end framework, where the learning can be performed straightforward via back-propagation. In contrast, for traditional codecs, some operations (e.g. quantization in JPEG) are not differentiable, so the model training via backpropagation now becomes non-traceable. To enable end-to-end optimization even when using the traditional codecs, we propose two strategies as described below.

Strategy I. The first strategy is to directly train an autoencoder-based simulation network to mimic the compression function via learning to reproduce the outputs of the traditional codec. Given a pair of uncompressed and compressed image $\left(I^H, I^L\right)$, where I^L is produced

by a specific traditional codec, the simulation network takes I^H as input and outputs a simulated compressed image I^{SL} via optimizing the the error between the I^{SL} and I^L.

Ideally, this pretrained simulation network could be considered as a differentiable version of the corresponding traditional codec, and should be capable of being smoothly integrated into our framework as the deep-learning-based codec. However, it is difficult to guarantee the approximation of any traditional codec by using the simulation network, as there exists quite a few highly-complex and nonlinear operations in the traditional codec. Therefore, there would be a discrepancy between the output of simulation network I^{SL} and the one obtained from the corresponding traditional codec I^L, in which it leads to problematic model training and even reduces the performance during testing.

To handle this issue, we propose a *skip-component*, which aims to keep the input signal of the decryption decoder D identical in both training and testing stages. To be specific, we design a pipeline which utilizes both simulation network and traditional codec during the training stage (as illustrated in the left of Fig. 2). Given an encoded image I^H_{enc}, the simulation network and the traditional codec output the corresponding compressed images I^{SL}_{enc} and I^L_{enc} respectively. The skip-component performs subtraction between I^L_{enc} and I^{SL}_{enc}, and add the difference $\nabla = I^L_{enc} - I^{SL}_{enc}$ back to I^{SL}_{enc}:

$$I^{SL}_{enc} + \nabla = I^{SL}_{enc} + (I^L_{enc} - I^{SL}_{enc}) = I^L_{enc}. \tag{1}$$

Based on this specific design of skip-component, the input for the decryption decoder D is always be I^L_{enc} no matter it is for training or testing. In other words, the decryption decoder D would learn to extract hidden information from the compressed image which is exactly produced by the traditional codec. Moreover, now the pretrained simulation network is no longer fixed, but needs to be updated via a loss function $\mathcal{L}_{sim}$ by using the error between I^{SL}_{enc} and I^L_{enc} since they are the actual pair compressed from I^H_{enc}:

$$\mathcal{L}_{sim} = \left\| I^{SL}_{enc} - I^L_{enc} \right\|_2^2. \tag{2}$$

Note that, when there are gradients backpropagated from the decryption side to the encryption side, they would flow along the differentiable simulation network instead of the traditional codec (as it is non-differentiable), as shown in Fig. 2. Despite that the skip-component provides a way to eliminate the influence of the discrepancy between output of the simulation network and the traditional codec, it still requires additional effort to train the simulation network. Therefore, we propose the second strategy to avoid training the simulation network.

Strategy II. Without the simulation network, we simplified the process and directly feed I^H_{enc} and I^L_{enc} into the skip-component (as illustrated in the right of Fig. 2), where now the difference between I^L_{enc} and I^H_{enc} is taken into consideration as:

$$\begin{aligned} \Delta &= I^L_{enc} - I^H_{enc}, \\ I^H_{enc} + \Delta &= I^H_{enc} + (I^L_{enc} - I^H_{enc}) = I^L_{enc}. \end{aligned} \tag{3}$$

In comparison to the Strategy I, the proposed Strategy II has the same advantage as decoder D always takes the input from I^L_{enc}, but is even more straightforward and simple, without requiring the simulation network anymore. However, we observe that the Strategy I can be further improved if the simulation network can better approximate the traditional codec, which is also an interesting research direction. Therefore, we study both strategies and compare their results together to provide insight from different perspectives for future work.

Despite the different design, during the testing stage, both Strategy I and Strategy II follow the same workflow sequentially as shown in Fig. 1: 1) Encoder E hides the residual I^R into the original image I^H and outputs the encoded image I^H_{enc}; 2) I^H_{enc} is fed into the compression codec to obtain its compressed version I^L_{enc}; 3) After receiving I^L_{enc}, the decoder D extracts the information hidden in I^L_{enc}, and then uses it to reconstruct the final output I'^H.

3.4 Objective Function

In this section, we describe the overall objective function used in our proposed framework.

Reconstruction Loss. Our goal is to recover the original image I^H in the end, so we adopt the reconstruction loss $\mathcal{L}_{rec}$ on the output of the decryption process, i.e. I'^H. Since our model is now end-to-end trainable, the loss applied on I'^H not only encourages the decryption decoder D to reconstruct the original image I^H, but also guides the encryption encoder E to properly encode the residual I^R according to the compression codec used in the compression proxy. The reconstruction loss $\mathcal{L}_{rec}$ is defined as:

$$\mathcal{L}_{rec} = \left\| I'^H - I^H \right\|_2^2. \tag{4}$$

Consistency Loss. In addition, we hope that the encoded image I^H_{enc} still well preserves the structure and content of the original image I^H since we are motivated by the idea of hiding residual, i.e. steganography. We hence have the consistency loss $\mathcal{L}_{con}$ as a regularization term, which encourages the encryption encoder E to hide the residual I^R in I^H without producing large deformation onto the image:

$$\mathcal{L}_{con} = \left\| I^H_{enc} - I^H \right\|_2^2. \tag{5}$$

Note that we do not further encourage I^L_{enc} to be close to I^L since we already utilize the above consistency loss to regularize I^H_{enc}, as the difference between I^H_{enc}/I^L_{enc} and I^H/I^L is made by the same compression codec. The overall objective $\mathcal{L}_{total}$ of our framework is defined as:

$$\mathcal{L}_{total} = \lambda \mathcal{L}_{rec} + \mathcal{L}_{con}, \tag{6}$$

where the hyper-parameter λ is the weight to balance two losses. Since $\mathcal{L}_{con}$ only acts as a regularization, we set a larger weight for λ equal to 5 in all the experiments. We use Adam [10] optimizer with learning rate of 10^{-4} and train our model for 100 epochs.

4 Experimental Results

Here we first introduce the datasets and the evaluation metrics utilized in our experiments. We then detail the settings of the compression codecs, and finally present quantitative and qualitative results of our method for boosting image compression, with comparisons to several baselines. All the source code and models will be made available to the public.

4.1 Datasets and Evaluation Metrics

Datasets. Two datasets are used in the experiments, Kinetics and Kodak. Kinetics is used for both training and testing, while Kodak is for testing only:

- Kinetics [9]: Kinetics is a high-quality and large-scale dataset collected from YouTube, which is composed of over 650K video clips about human activities in reality. We sample two non-overlapping subsets of Kinetics for training and testing, respectively. The training set consists of 10,128 images in 1093 videos over 391 classes, and the testing one contains 3253 images in 345 videos over 345 classes.

- Kodak [11]: We also use the Kodak dataset only for the testing purpose. Kodak consists of 24 lossless images and is commonly used for evaluating image compression.

Metrics. We adopt PSNR and SSIM [18] as our quantitative metrics, which are commonly used in most research works on image compression.

4.2 Experimental Settings

We conduct experiments on several traditional image coding standards (JPEG, JPEG2000, and BPG) as well as the deep-learning-based codecs (CAE-B, CAE-P, and RNN) under two types of evaluation settings. Two baseline models are used for making the comparison: (1) an artifact-removal network DnCNN [19] which is widely used to reduce the distortions caused by compression, and (2) an image restoration model MemNet [14] which performs restoration (e.g. JPEG de-blocking) on the Y channel in the YCbCr color space. We also provide comparisons of our model to an end-to-end compression framework [8] in the supplementary material as its experimental setting is different from ours.

Compression Codecs. There are three deep-learning-based compression codecs: CAE-B [20], CAE-P [20], RNN [15], and three traditional based codecs: JPEG, JPEG2000, and BPG, used in our experiments. Since the coding of each codec is very different, we categorize them into two groups: *stable group* and *dynamic group*. **Stable group** contains CAE-B, RNN, and JPEG2000, in which the bpp (bit per pixel) is the same when the compression quality is fixed. **Dynamic group** consists of CAE-P, JPEG, and BPG, in which the bpp varies under a fixed compression quality. The varying bpp is caused by the coding algorithm inside the codec, such as entropy coding. Different images would lead to different bpp according to the structure and details of the image content. Please note that since our model would somewhat change the details (by hiding residual) of the original image, the bpp of the compressed encoded image (i.e. I^L_{enc}) increases a bit.

Simulation Network. As mentioned in Section 3, we have tried to simulate the functionality of traditional codecs by using deep models. For JPEG, since its function block is simple to implement, we follow the noise layer in HiDDeN [21] to approximate JPEG. Instead of using the mask as proposed in [21], we re-implement the quantization operation of JPEG and use the skip-connection technique to ensure that backpropagation can work smoothly. For JPEG2000 and BPG, we adopt an autoencoder-based architecture composed of 8 conv-Relu blocks without down-sampling on the image size.

4.2.1 Training Settings.

Training Settings. As our baselines are post-processing methods, the bpp variation should be considered for those experiments based on the codecs in the dynamic group, in order to have fair comparisons. We conduct two types of evaluation to compare our method with baselines fairly:

– Fixed-quality, denoted as **Fix-Q**, is to train our model under a fixed quality (e.g. 50 for JPEG) of codecs and test on that quality (i.e. 50) as well. However, our model would

Table 1: Results on the Kinetics (left) and Kodak (right) datasets with several deep-learning-based codecs (i.e., CAE-B, CAE-P, and RNN). Bold number indicates the best and underlined number indicates the second best performance.

	Dataset	Kinetics						Kodak					
	Coding Standard	CAE-B		CAE-P		RNN		CAE-B		CAE-P		RNN	
	Bpp (bits/pixel)	0.5	2.0	0.8	2.5	0.5	1.0	0.5	2.0	0.8	2.5	0.5	1.0
PSNR	Original	26.162	29.835	30.978	34.108	27.990	30.822	25.494	28.774	29.677	32.797	26.890	29.512
	MemNet	26.774	30.586	31.492	35.136	28.624	31.635	25.774	29.019	29.785	30.133	27.354	29.949
	DnCNN	26.805	30.520	31.599	35.511	28.459	31.403	25.955	29.197	30.069	33.652	27.337	29.963
	Ours	**27.258**	**31.584**	**31.911**	**35.875**	**28.935**	**31.914**	**26.636**	**30.441**	**30.584**	**34.799**	**27.761**	**30.337**
SSIM	Original	0.799	0.892	0.924	0.965	0.851	0.914	0.740	0.858	0.888	0.953	0.769	0.861
	MemNet	0.823	0.908	0.930	0.970	**0.868**	0.924	0.752	0.865	0.889	0.915	0.788	0.869
	DnCNN	0.822	0.908	0.932	0.971	0.864	0.922	0.754	0.869	0.891	0.957	0.785	0.869
	Ours	**0.831**	**0.924**	**0.933**	**0.972**	**0.868**	**0.926**	**0.769**	**0.888**	**0.892**	**0.960**	**0.791**	**0.874**

slightly create additional bpp when the codec belongs to the dynamic group. To have a fair comparison with those artifact-removal baselines, we train and test these baselines under a higher quality (e.g. 62 for JPEG) such that the average bpp of the compressed images is similar to our compressed case (encoded images I^L_{enc}).

– Fixed-bpp, denoted as **Fix-BPP**, is to train both our model and the baselines under a fixed quality (e.g. 50 for JPEG), and then test them under a fixed bpp. We set the bpp of a compressed encoded image I^L_{enc} in our model to a fixed value, like 1.0 for JPEG. Therefore, we actually input the corresponding compressed image I^L with a lower bpp, i.e., 0.889 here for example. Under this setting, we could make sure both the training and testing data are the same for our model and baselines.

4.3 Quantitative Results

Deep-learning-based Codecs. We evaluate the efficacy of our model of boosting image compression on several deep-learning-based codecs (CAE-B, CAE-P, and RNN) and make comparisons with the baselines, as shown in Table 1. The bpp of images compressed by CAE-B and RNN is fixed, and the value is shown in the tables. For CAE-P which belongs to the dynamic group, we discover that the bpp of our compressed encoded image (2.527 on average) is very close to the compressed one (2.535 on average). Therefore, we confirm that all the deep-learning-based codec used in our experiments need not to be concerned with the bpp issue. We can observe from the quantitative results provided in Table 1, that our proposed model outperforms the baselines by a significant margin in terms of PSNR and SSIM. Moreover, the results on Kodak shown in Table 1 indicate that our method can be well generalized to the unseen dataset.

Traditional Codecs. Next, we conduct evaluation on the traditional codecs (JPEG, JPEG2000, and BPG) in order to show our generalizability on different codecs in Table 2. Both JPEG and BPG are in the dynamic group so that we evaluate our model under both the fixed-quality and fixed-bpp settings, while JPEG2000 is with fixed bpp for a given quality so that we only consider the fixed-bpp setting. In addition, as mentioned in Section 3.3, we have two strategies to handle the non-differentiable property of traditional codec, and we evaluate our models with both strategies, denoted as **Ours-Strat. I** and **Ours-Strat. II** respectively in Table 2. From the results, Ours-Strat. II has a great boost in performance, and performs consistently better than other baselines (MemNet and DnCNN) on both the Kinetics Kodak datasets under all the codecs and settings. It shows that our proposed model indeed encodes the residual into the original image and tries to recover it with the useful hidden information.

Table 2: Results on the Kinetics (left) and Kodak (right) datasets with several traditional coding standards (i.e., BPG, JPEG, and JPEG2000). Bold number indicates the best and underlined number indicates the second best performance.

	Dataset	Kinetics					Kodak				
	Coding	BPG		JPEG		JPEG2000	BPG		JPEG		JPEG2000
	Mode	Fix-Q	Fix-BPP	Fix-Q	Fix-BPP	Fix-Q/BPP	Fix-Q	Fix-BPP	Fix-Q	Fix-BPP	Fix-Q/BPP
	Quality / Bpp	30 / 0.702	- / 1.0	50 / 0.889	- / 1.0	48 / 0.50	30 / 0.702	- / 1.0	50 / 0.889	- / 1.0	48 / 0.50
PSNR	Original	36.404	39.271	32.640	33.826	29.292	35.373	37.084	32.059	32.749	28.575
	MemNet	38.217	39.111	35.095	34.883	30.295	37.251	37.002	34.148	33.340	28.987
	DnCNN	39.073	39.208	35.783	35.536	30.465	37.758	37.204	34.258	33.889	29.413
	Ours-Strat. I	39.028	**40.009**	36.359	**36.126**	30.313	37.842	37.555	34.958	**34.273**	29.300
	Ours-Strat. II	**39.099**	39.989	**36.370**	36.121	**30.945**	**37.917**	**37.578**	**34.960**	34.247	**30.108**
SSIM	Original	0.967	0.965	0.919	0.923	0.818	0.951	0.944	0.918	0.903	0.754
	MemNet	0.967	0.965	0.947	0.937	0.841	0.951	0.944	0.925	0.911	0.768
	DnCNN	0.972	0.968	0.957	0.947	0.847	0.955	0.947	0.934	0.918	0.778
	Ours-Strat. I	0.974	**0.972**	**0.963**	**0.953**	0.857	0.960	0.953	**0.940**	**0.927**	0.795
	Ours-Strat. II	**0.975**	**0.972**	0.962	0.952	**0.874**	**0.961**	**0.955**	**0.940**	**0.927**	**0.826**

Moreover, we observe that our model with the simulation network (i.e. Ours-Strat. I) has comparable performance with Ours-Strat. II on JPEG and BPG, but has worse results on JPEG2000. This difference shows that the simulation network may cause our model fail to learn effectively to hide the residual if it can not approximate the real codec well.

Figure 3: Qualitative results on the Kodak dataset. Some details are highlighted with the zoom-in patches. We show that our proposed method produces clearer reconstruction with more details and textures. For example, the wool on the red scarf shown in column (e) of the last row is clearer than the others.

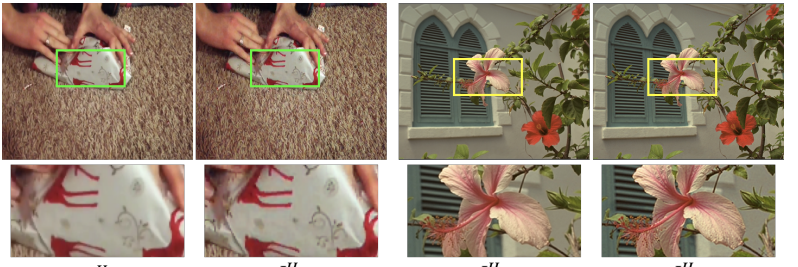

Figure 4: Example results for visualizing the original image I^H and the encoded uncompressed one I^H_{enc}. For each example, the image on the left is I^H and the image on the right is I^H_{enc}. We observe that I^H_{enc} is with more sharpened details.

4.4 Qualitative Results

We provide some examples of original images I^H, compressed images I^L, results from baselines and our model, in Fig. 3. These results demonstrate that our model is able to recover more image details than baselines. Especially in some cases (e.g., the details on the red scaff in Fig. 3), both MemNet and DnCNN produce artifacts of having blurs, while the results from our model still provide sharper details. Due to the page limit, we provide more qualitative results in the supplementary materials.

To further understand how our *hiding* scheme works, we visualize our encoded image I^H_{enc} before compression happens, i.e., encoding I^H with the residual I^R. In Fig. 4, we show comparisons of I^H and I^H_{enc}. Interestingly, I^H_{enc} looks sharper than the original image I^H, where the detailed textures are enhanced. With this observation, we could treat our *hiding* process as a *pre-amplified* step on the original image through the encoder E, as the model knows that there is an afterward compression process which could remove some details. Therefore, with the enhancement on those details via adding hidden residual before compression, the decoder D receives a more informative image and is capable of reconstructing a better output.

5 Conclusions

In this paper, we propose a framework that learns to hide the residual information during the image compression process, for reducing the artifacts caused by compression. To this end, we construct encryption and decryption processes, in which the former one encodes/hides the residual into the original image for transmission, and the latter one decodes the residual back on the receiver side for boosting the output quality. In addition, our method is end-to-end trainable and can be integrated into existing deep-learning-based codecs. For traditional codecs such as JPEG, JPEG2000, and BPG, we develop two strategies to allow our model applicable to these codecs, while still achieving promising results. In experiments, we provide extensive comparisons to existing artifact-removal methods under various image compression codecs, and show the efficacy of the proposed framework via hiding the residual.

6 Acknowledgement

This project is supported by Qualcomm technologies, Inc. (NAT-439543), MOST 110-2636-E-009-001, and MOST 110-2634-F-009-023, Taiwan. We are grateful to the National Center for High-performance Computing, Taiwan, for computer time and facilities.

References

[1] Tinku Acharya and Ping-Sing Tsai. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. 2004.

[2] 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 (ICLR)*, 2018.

[3] Fabrice Bellard. BPG Image Format. http://bellard.org/bpg/, 2014.

[4] Huibin Chang, Michael K Ng, and Tieyong Zeng. Reducing artifacts in jpeg decompression via a learned dictionary. *IEEE Transactions on Signal Processing (TSP)*, 2013.

[5] Tao Chen, Hong Ren Wu, and Bin Qiu. Adaptive postfiltering of transform coefficients for the reduction of blocking artifacts. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2001.

[6] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision (ECCV)*, 2014.

[7] Chao Dong, Yubin Deng, Chen Change Loy, and Xiaoou Tang. Compression artifacts reduction by a deep convolutional network. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[8] Feng Jiang, Wen Tao, Shaohui Liu, Jie Ren, Xun Guo, and Debin Zhao. An end-to-end compression framework based on convolutional neural networks. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2017.

[9] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *ArXiv:1705.06950*, 2017.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv:1412.6980*, 2014.

[11] Eastman Kodak. Kodak PhotoCD dataset. http://r0k.us/graphics/kodak/, 1993.

[12] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[13] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[14] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[15] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] Yi-Hsuan Tsai, Ming-Yu Liu, Deqing Sun, Ming-Hsuan Yang, and Jan Kautz. Learning binary residual representations for domain-specific video streaming. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[17] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 1991.

[18] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing (TIP)*, 2004.

[19] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. In *IEEE Transactions on Image Processing (TIP)*, 2017.

[20] Haimeng Zhao. Cae-p: Compressive autoencoder with pruning based on admm. *ArXiv:1901.07196*, 2019.

[21] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *European Conference on Computer Vision (ECCV)*, 2018.